TECHNISCHE
UNIVERSITÄT
DARMSTADT

PERFORMANCE AND AVAILABILITY IN PEER-TO-PEER CONTENT DISTRIBUTION SYSTEMS: A CASE FOR A MULTILATERAL INCENTIVE APPROACH

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertation

von

DIPL.-INFORM. SEBASTIAN KAUNE

Geboren am 14. November 1979 in Darmstadt, Hessen

Vorsitz: Prof. Dr.-Ing. Thomas Hartkopf

Referent: Prof. Dr.-Ing. Ralf Steinmetz
Korreferent: Prof. Dr.-Ing. Jörg Eberspächer
Korreferent: Dr. Andreas Mauthe

Tag der Einreichung: 25. November 2010
Tag der Disputation: 10. Februar 2011

Hochschulkennziffer D17
Darmstadt 2011

ABSTRACT

The peer-to-peer paradigm offers the potential to address many of the challenges related to large-scale content distribution over the Internet (e.g., scalability, costs, etc.). However, currently, many approaches fail to offer the quality of service supported by over-provisioned client-server distribution systems such as RapidShare and YouTube. The reason for this is that peer-to-peer systems must operate under the constraints placed on them by contributing users. Consequently, selfish, strategic, or malicious users often lower the overall system utility. To address this, *incentive mechanisms* are employed to motivate such users to cooperate with the system (e.g., those that contribute more, receive more in return). Many users, however, observe that popular systems such as BitTorrent (employing tit-for-tat as incentive mechanism), are often ineffective at fulfilling a set of key content distribution requirements, namely *performance* and *availability*. The overarching goal of this thesis is therefore to extend existing incentive mechanisms to better fulfil these core requirements.

At first, this thesis validates and quantifies these casual observations regarding peer-to-peer performance and content availability. To achieve this we carry out two major measurement studies in the BitTorrent system, the current de-facto standard for peer-to-peer content distribution. To this end we identify widespread performance and availability problems that, through detailed analysis, are attributed to *ineffective incentive design*. In particular, we find that the current popular approach of 'tit-for-tat' fails to incite sufficient cooperation amongst users to ensure high performance and content availability. Further analysis shows that this is caused by a lack of incentives for *seeding* (i.e., the process of remaining as a data source after one has downloaded the entire file).

Based on these findings, we subsequently study a set of intuitive solutions to overcome what we call the *seeder promotion problem*. The purpose of this analysis is to narrow the solution space and to shape a more sophisticated incentive design. In order to achieve this, three abstract cross-torrent incentive approaches are detailed, as well as single-torrent incentive mechanisms. Each of these approaches are then quantitatively analysed through extensive trace-based simulations. These analysis further confirm our finding that bilateral incentive strategies (e.g., tit-for-tat) are insufficient at providing robust incentives for seeders. This is because most users *(i)* do not meet each other repeatedly and *(ii)* do not simultaneously require each other's content. Instead, it is shown that the only way to overcome performance and availability issues is to use *multilateral* incentive strategies (i.e., to allow users to contribute to one user yet receive reciprocation from another).

Finally, we design and evaluate a novel multilateral incentive mechanism, named *FairSwarm.KOM*. Unlike digital currency systems (in which contribution information is globally visible) or tit for-tat (where no propagation of credit points occurs), FairSwarm.KOM uses one-hop information of the overlay network to evaluate the cooperativeness of the peers. Through the use of extensive trace-based simulations, it is shown that FairSwarm.KOM improves the download performance of the popular BitTorrent system by more than 86%, while guaranteeing high levels of file availability (>99%). Most importantly, these two properties are achieved without harming the fairness of individual users and with an extremely low overhead.

KURZFASSUNG

Das Peer-to-Peer (P2P)-Kommunikations-Paradigma bietet neue Möglichkeiten für die Verteilung von digitalen Inhalten über das Internet. Peer-to-Peer bündelt die Ressourcen der Teilnehmer im System und setzt diese effizient ein, um eine Datei an eine große Anzahl von interessierten Nutzern zu verteilen. Damit sind P2P-Systeme inhärent selbst-skalierend: Je mehr Teilnehmer dem System beitreten, desto höher die Anzahl der verfügbaren Ressourcen und desto besser die Verteilungsgeschwindigkeit. Jedoch profitieren nicht nur die Nutzer von P2P-Technologie; die Anbieter digitaler Medien (Content-Provider) können gerade durch die Nutzung dieser Ressourcen ebenfalls Upload-Kosten durch den Quellserver einsparen.

Trotz dieser beachtlichen Vorteile gegenüber traditionellen Client/Server-Ansätzen können bestehende P2P-Inhaltsverteilsysteme, wie zum Beispiel das populäre BitTorrent-System, bezüglich der Downloadgeschwindigkeit und der Langzeitverfügbarkeit von Inhalten oftmals nicht mit ressourcenstarken Client/Server-Farmen wie RapidShare oder YouTube konkurrieren. Dies ist insbesondere darauf zurückzuführen, dass die Leistungsfähigkeit eines Peer-to-Peer-Systems einzig und allein auf der Kooperationsbereitschaft der teilnehmenden Nutzer beruht. Jedoch neigen gerade die Endnutzer dazu, ihre Ressourcenbereitstellung bis auf ein Minimum zu reduzieren, solange kein *Anreiz* zur Kooperation besteht. Angesichts dieses Dilemma und der häufig zu beobachtenden Effizienzprobleme von P2P-Inhaltsverteilsystemen, ist daher zu vermuten, dass existierende Anreizmechanismen zur Erhöhung von Kooperation, wie z. B. der vorwiegend eingesetzte „Tit-for-Tat"-Mechanismus in BitTorrent, ihre Rolle gänzlich verfehlen. Genau an diesem Punkt setzt die vorliegende Dissertation an; dem Entwurf und der Konzeptionierung eines neuartigen Anreizmechanismus zur Förderung von *nachhaltiger* Kooperation, der sowohl höhere Downloadgeschwindigkeiten erzielen soll, als auch die Langzeitverfügbarkeit von Inhalten in P2P-Systemen verbessert.

Um die Korrektheit der oben genannten Schlussfolgerungen wissenschaftlich zu untermauern, beschäftigt sich diese Arbeit zunächst mit der Validierung, Quantifizierung und Ursachenanalyse bestehender Effizienzprobleme in P2P-Inhaltsverteilsystemen. In diesem Kontext werden zwei Messstudien im populären BitTorrent-System durchgeführt, das den gegenwärtigen de-facto-Standard für moderne P2P-Inhaltsverteilung repräsentiert. Hierbei wird gezeigt, dass P2P-Systeme sowohl von gravierenden Geschwindigkeits- als auch Verfügbarkeitsproblemen betroffen sind. Durch die Messdatenanalyse kann des Weiteren belegt werden, dass diese Effizienzprobleme auf falsch gesetzte Anreize zur Kooperation zurückzuführen sind. Genauer betrachtet schafft es der Tit-for-Tat-Mechanismus nicht, Nutzern Anreize zur Kooperation zu geben, wenn diese ihren Dateidownload beendet haben. Jedoch sind genau diese Datenquellen, die so genannten „Seeder", ausschlaggebend für die Downloadgeschwindigkeit und Verfügbarkeit von Inhalten in P2P-Systemen.

Basierend auf diesen Erkenntnissen wird anschließend eine eingehende Analyse von möglichen Lösungsansätzen zur Schaffung von Anreizen für Seeder durchgeführt. In Folge dessen wird der mögliche Lösungsraum weiter eingegrenzt und es können grundlegende Lösungsanforderungen ermittelt werden. In diesem Zusammenhang werden sowohl drei „Torrent"-übergreifende Lösungsansätze als auch „Single-Torrent"-basierte Ansätze durch messdatengestützte Simulationen quantitativ untersucht. Hierbei bestätigt sich, dass bilaterale Anreizstrategien (wie z. B. Tit-for-Tat) gänzlich ungeeignet sind, um nachhaltige Kooperationen in P2P-Systemen zu fördern. Dies ist dadurch begründet, dass Nutzer sich oftmals nur einmal im P2P-System begegnen und keine Tauschgrundlage bezüglich bereits heruntergeladener Inhalte besitzen. Stattdessen kann aufgezeigt werden, dass ausschließlich *multilaterale* Anreizmechanismen das

Potenzial aufweisen, langanhaltende Anreize für Seeder zu schaffen, und somit der Schlüssel zur Lösung der oben beschriebenen Effizienzprobleme sind. Mithilfe eines multilateralen Anreizmechanismus kann ein Nutzer beispielsweise Daten zu einem beliebigen Teilnehmer im System transferieren, um diese Kooperation dann von einem beliebigen Dritten (zu einem späteren Zeitpunkt) zurückzufordern.

Anhand der ermittelten Anforderungen für eine konkrete Lösung beschäftigt sich der letzte Teil dieser Arbeit mit dem Entwurf und der Konzeptionierung eines neuartigen Anreizmechanismus namens „FairSwarm.KOM“. Im Gegensatz zu bestehenden Tit-for-Tat-Mechanismen, in denen die Nutzer keine Informationen über geschehene Kooperationen austauschen, und Anreizmechanismen, die jede Kooperation eines Teilnehmers im System öffentlich sichtbar machen (Virtual Currency), verwendet FairSwarm.KOM „Ein-Hop“-Nachbarschaften, um die Kooperationsbereitschaft einzelner Teilnehmer zu ermitteln. Es wird durch messdatengestützte Simulationen gezeigt, dass dieser Ansatz eine nahezu 100-prozentige Dateiverfügbarkeit in BitTorrent-Systemen garantiert und auch die Downloadgeschwindigkeit im Vergleich zu Tit-for-Tat-Mechanismen um mehr als 86 % verbessern kann. Zudem werden diese Eigenschaften erreicht, ohne dabei die Fairness-Grundlagen des Systems zu verletzen und ohne einen signifikanten Kommunikationsoverhead im System zu erzeugen.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Prof. Dr.-Ing. Ralf Steinmetz, for giving me the opportunity to become a doctor at the Multimedia Communications Lab and for allowing me to follow my own research path. I am also very grateful for his constant support and guidance during my PhD. Also, I would like to thank my co-supervisor, Dr. Andreas Mauthe, for all the useful advice and encouragement at every stage of my work. I am also very grateful to Prof. Dr.-Ing. Jörg Eberspächer for his help, especially while finishing this work.

Special thanks goes to Dr. Gareth Tyson for his precious help and support during my entire thesis work. It is always a pleasure to work with you! I also thank Dr. Rubén Cuevas Rumín for all the constructive discussions we had over the years.

I am very grateful to my colleagues from the "Peer-to-Peer System" research group. In particular, special thanks to Konstantin and Sandra for helping me out in so many situations during my work at KOM. Also, I want to thank Osama, Patrick, Christian and Dominik for their valuable comments at the end of this work.

Furthermore, many thanks go to the entire KOM team, especially to Julian, Stefan, André, Karola, Sabine, Moni, Frau Scholz-Schmidt and Frau Ehlhardt.

Last, I am deeply thankful to my parents, Wolfgang and Monika, and my sister Vanessa for their love and support during my entire life! Last but not least, I thank Manuela for her being such a compassionate, loving and wonderful person.

Darmstadt 2011

CONTENTS

1 INTRODUCTION 1
- 1.1 Motivation . . . 1
- 1.2 Incentive Mechanisms . . . 2
- 1.3 Research Challenges . . . 3
- 1.4 Research Goals . . . 4
- 1.5 Thesis Organisation . . . 4

2 BACKGROUND AND RELATED WORK 7
- 2.1 Introduction . . . 7
- 2.2 Basic Principles of P2P Content Distribution . . . 7
  - 2.2.1 Overview . . . 7
  - 2.2.2 Discovery Mechanisms . . . 8
  - 2.2.3 Delivery Mechanisms . . . 10
- 2.3 Widely Deployed P2P Content Distribution Systems . . . 13
  - 2.3.1 Gnutella . . . 13
  - 2.3.2 eDonkey/eMule . . . 13
  - 2.3.3 BitTorrent . . . 14
- 2.4 Related Work . . . 16
  - 2.4.1 Overview . . . 16
  - 2.4.2 Direct Reciprocity . . . 17
  - 2.4.3 Indirect Reciprocity . . . 19
  - 2.4.4 Virtual Currency . . . 20
- 2.5 Conclusions . . . 21

3 MEASUREMENTS AND ANALYSIS OF THE BITTORRENT SYSTEM 23
- 3.1 Introduction . . . 23
- 3.2 Measurement Methodology . . . 24
  - 3.2.1 Overview . . . 24
  - 3.2.2 Microscopic Crawling . . . 25
  - 3.2.3 Macroscopic Crawling . . . 27
- 3.3 Measuring User-Level Issues in the BitTorrent System . . . 29
  - 3.3.1 Download Throughput . . . 29
  - 3.3.2 Abortion Rates . . . 30
- 3.4 Causes of Limitations: Performance and Availability . . . 31
  - 3.4.1 The Role of Seeders in Download Performance . . . 31
  - 3.4.2 The Role of Seeders in File Availability . . . 33
  - 3.4.3 The Emergence of Seedless States . . . 36
- 3.5 Scale of Limitations . . . 39
  - 3.5.1 The Prevalence of Seedless States . . . 40
  - 3.5.2 The Extent of File Unavailability . . . 41
- 3.6 Conclusions . . . 41

4 ANALYSIS OF SOLUTION SPACE 43
- 4.1 Introduction . . . 43
- 4.2 Abstract Solution Space . . . 44
  - 4.2.1 Single-Torrent Approaches . . . 44
  - 4.2.2 Cross-Torrent Approaches . . . 45
  - 4.2.3 Summary . . . 47
- 4.3 Quantitative Analysis of Abstract Solution Space . . . 48
  - 4.3.1 Experimental Methodology . . . 48

4.3.2 Formative Results . . . 51
4.3.3 Summary . . . 55
4.4 Concrete Solution Requirements . . . 56
4.4.1 Multilateral Incentive Design . . . 56
4.4.2 Decentralisation . . . 56
4.4.3 Robustness . . . 57
4.4.4 Contribution Fairness . . . 58
4.5 Conclusions . . . 58

5 A NEW P2P INCENTIVE MECHANISM – FAIRSWARM.KOM 61
5.1 Introduction . . . 61
5.2 Conceptual Overview . . . 61
5.3 Coupons . . . 63
5.3.1 Coupon Data Fields . . . 63
5.3.2 Coupon Signatures . . . 64
5.4 Coupon Propagation Specification . . . 65
5.4.1 Selection of Indirection Level . . . 65
5.4.2 Bootstrapping of Indirect Reciprocation . . . 66
5.4.3 Maintaining Indirect Reciprocation . . . 68
5.4.4 Managing Coupon Versions . . . 69
5.5 Data Exchange Specification . . . 70
5.5.1 Requirements of Specification . . . 70
5.5.2 Specification Overview . . . 72
5.5.3 Protocol Notations . . . 73
5.5.4 Exchange Protocol . . . 75
5.5.5 Abort Protocol . . . 76
5.5.6 Recovery Protocol . . . 77
5.5.7 Complaint Protocol . . . 78
5.5.8 Summary . . . 79
5.6 Service Policies . . . 79
5.6.1 Default Policy . . . 80
5.6.2 Alternative Policies . . . 81
5.6.3 Discussion . . . 83
5.7 Implementing FairSwarm.KOM in BitTorrent . . . 84
5.7.1 Identity Management . . . 84
5.7.2 Unchoking . . . 85
5.7.3 Propagation of Coupons . . . 86
5.8 Solution Requirements Revisited . . . 86
5.9 Conclusions . . . 88

6 EVALUATION OF FAIRSWARM.KOM 91
6.1 Introduction . . . 91
6.1.1 Evaluative Aims . . . 91
6.1.2 Evaluation Methodology . . . 91
6.2 System Parameter Tuning . . . 92
6.2.1 Experimental Methodology . . . 92
6.2.2 Percentage Threshold for Coupon Updates . . . 93
6.2.3 Number of Probabilistic Coupons . . . 95
6.3 Comparing P2P Incentive Mechanisms . . . 97
6.3.1 Experimental Methodology . . . 97
6.3.2 Availability . . . 99
6.3.3 Performance . . . 100
6.3.4 Fairness . . . 101
6.3.5 Overhead . . . 103
6.4 Robustness to Untruthful User Behaviour . . . 105

6.4.1 Experimental Methodology . . . 105
6.4.2 Free-Riding and Whitewashing . . . 106
6.4.3 Sybil Attack with Fixed Identities . . . 109
6.4.4 Sybil Attack with Changing Identities . . . 111
6.5 Evaluative Summary . . . 114

7 CONCLUSION 117
7.1 Summary of Thesis . . . 117
7.2 Major Contributions . . . 119
7.3 Other Contributions . . . 120
7.4 Research Goals Revisited . . . 121
7.5 Future Work . . . 122
7.6 Concluding Remarks . . . 123

REFERENCES 125

A APPENDIX 135
A.1 Session Time Estimation . . . 135
A.2 Investigating the Causes of Swarm Resilience . . . 135
A.3 Security Analysis of Data Exchange Specification . . . 137
A.4 Analysing User Interaction Pattern . . . 139
A.5 Accuracy of the Simulation Model of BitTorrent . . . 141

B AUTHOR'S PUBLICATIONS 147
B.1 Main Publications . . . 147
B.2 Other Publications . . . 147

C CURRICULUM VITAE 149

D ERKLÄRUNG LAUT §9 DER PROMOTIONSORDNUNG 151

1 INTRODUCTION

Content distribution over the Internet is becoming increasingly popular amongst both the industry and consumers alike. Recently, a market analysis has shown that 25% of all music purchased in the U.S. is downloaded from Apple iTunes and it is expected that digital music sales will nearly equal CD sales by the end of 2011 [6]. This trend is confirmed by the observation that a large number of key content producers including MGM, Universal, Disney, and 20th Century Fox are now distributing films and television shows via the Internet [86]. These online stores offer millions of users a simple way to access a wide variety of pre-stored content while remaining as a high profit-yielding business model.

During the last years, not only the number of Internet users has experienced an explosive growth, but also the size of content being distributed. Videos are available in higher resolutions so that a 90-minute movie in high-definition quality already exceeds 2 GByte. In addition to this, many users have access to resident broadband networks such as Digital Subscriber Lines (DSL) and cable connections. More than 283 million people already use these networks worldwide [36], and this number is expected to rise to 477 million by 2011 [25]. As a consequence, content providers need to be prepared to handle potentially hundreds of thousands of well-connected users simultaneously accessing a single file object amounting to several gigabytes.

Not surprisingly, source servers are recently often crowded and suffer from link congestion. For instance, the beta release of Microsoft Vista and Windows 7 were restricted to only a few thousands downloads, but still the source servers could not withstand these peak demands and suddenly crashed [119, 91]. These issues are often addressed by hardware upgrades (i.e., adding additional servers) to increase the bandwidth available and/or the amount of parallel connections that can be maintained. This is, however, only a temporary solution as it does not tackle the problem while it occurs; there can always appear more users necessitating additional upload resources. Consequently, a solution also requires that the system can spontaneously adapt to the unpredictable and continuously changing demand of users.

1.1 MOTIVATION

Peer-to-peer (P2P) networks have a great potential to address these long-standing issues. While early P2P research has focused almost exclusively on content localisation [18, 41, 104, 117], one of its most promising applications is *content distribution* [73, 4].

In P2P content distribution, users (synonymously called peers or nodes) are both suppliers and consumers of bandwidth, in contrast to the traditional client-server model where only servers supply and clients consume. When aggregated by a P2P network, these nodes constitute an immense pool of under-utilised bandwidth that can be leveraged by protocols such as BitTorrent to quickly distribute content files over the Internet [21]. As shown in Figure 1, in BitTorrent, the file to be distributed is typically split into many chunks that are directly exchanged between the interested peers. Since peers not only download chunks from users in possession of the file (e.g., source servers), but also serve them to other users, the serving capacity of the system increases with the number of available nodes, making the system potentially self-scaling.

From the content provider's perspective, P2P content distribution offers two potential benefits. First, expensive bandwidth over-provisioning to withstand peak-demands is negated to the point of irrelevance, because the system automatically adapts to its needs. Second, the content provider can forbear from purchasing services of third-

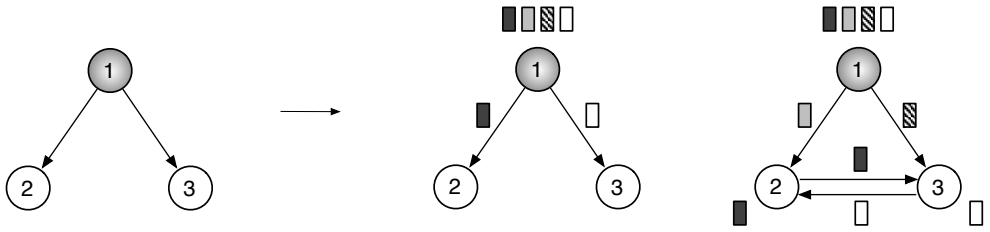


Figure 1: Modern peer-to-peer content distribution systems split large files into many chunks. Source node 1 simultaneously uploads file chunks to receivers 2 and 3. Both receivers then immediately start to mutually exchange already received chunks.

parties such as Akamai. This cost-effectiveness is further validated by the fact that leading content producers such as MTV, Paramount Pictures and 20th Century Fox have recently partnered BitTorrent Inc. [13]. Also, leading players in the gaming industry, e.g., Blizzard Entertainment, already use the BitTorrent protocol to distribute important patches and software updates for their famous online game 'World of Warcraft' [14].

When considering the above, P2P content distribution seems to have many advantageous characteristics compared to client-server based approaches for both content providers and end-users. Although P2P technologies have been applied to commercial environments, their predominant usage remains in the domain of *open systems*. These are systems that do not maintain strict control over peer and protocol behaviour. Instead, they are systems that allow open access to any user willing to cooperate. Consequently, through the use of modified source code, it becomes possible for any peer to deviate from the protocol specification at its own will. Such systems already account for 50-70% of the total Internet share [49]. Recent work has shown that these systems can be used in a commercial context [75, 76, 46]; however, issues related to the *continuous provision of content* have so far not been sufficiently addressed.

In particular, measurement studies such as [98, 94, 5] demonstrate that in P2P content distribution systems[1], users often have to wait long periods of time before being able to finish their downloads. Unfortunately, however, client *download performance* is a major indicator of user satisfaction, and thus the success of the entire system. Users simply expect that the system fully utilises the downstream bandwidth of their broadband connections. While this problem is often mitigated when accessing popular content early in its lifecycle, significant performance issues can be observed in P2P systems distributing older, less popular items. Specifically, this content not only suffers from low performance but also from long-term *unavailability* [42, 84]. This is, however, a highly unsatisfactory situation; ideally, it should be guaranteed that users can download the entire range of pre-stored content including popular as well as unpopular file objects. Especially the latter are of particular importance, since unpopular file objects can represent a significant fraction of demand and revenue [3].

This thesis primary focuses on these *performance* and *availability* issues in (open platform) P2P content distribution systems.

1.2 INCENTIVE MECHANISMS

A number of potential causes can be identified when inspecting performance and availability issues in P2P content distribution. Most notable is the poor resource availability that can be observed at some peers. Traditionally, this has been used to explain why not the same level of performance can be achieved compared to over-provisioned client-server systems. Recent studies, however, have identified that resource availability is *not* the core reason [94]. Instead, the willingness of peers to contribute

1 When we use the term P2P content distribution system in the remainder of this thesis, we implicitly refer to open platform systems.

their resources is, in fact, the true cause [48]. Thus, the resources are sufficient but the users are simply not prepared to contribute them, meaning that the solution to these performance and availability issues should be oriented towards incentivising users to better cooperate with the system. This is most systemic in systems such as Gnutella that do not provide any incentives at all.

From this it is clear that a logical approach to resolve the issues inspected in this thesis is to improve the incentive mechanisms employed by P2P content distribution systems. Subsequently, the purpose of this would be to ensure that strategic (selfish) users are always encouraged to contribute their resources through the promise of rewards. Recent years have seen a set of similar efforts taking place, with the predominant technique focussing on 'tit-for-tat'-policies [7], i.e., node A will only upload to node B if node B will upload to node A. This approach, however, has failed to address performance and availability issues, as validated later in this thesis. Consequently, the next stage must be to investigate the research challenges related to the design of superior incentive schemes.

1.3 RESEARCH CHALLENGES

Designing incentives to fully exploit the potential of P2P content distribution is complex and highly challenging. In particular, due to the constraints of the environment that P2P content distribution systems operate in (i.e., the Internet), a sophisticated design has to consider many characteristics at the same time, viz.:

Challenge 1: *Lack of Centralised Control.*

Incentive design becomes trivial if we can assume a central authority in P2P systems that immediately detects and punishes uncooperativeness while keeping account of user contributions. P2P networks are, however, assumed to operate in a completely decentralised fashion without a global 'police' that governs interactions between the nodes [116, 81]. As a result, users are free to attempt to strategically manipulate others, facilitated by the impossibility of peer interactions being monitored by third-parties. This makes incentive design extremely challenging because it must be devised such that it motivates even selfish users to cooperate, without having to resort to a centralised authority. Consequently, such a mechanism must be *robust* to attacks and misbehaving users.

Challenge 2: *Need for Continuity of Data Sources.*

In a P2P system, inciting users that are downloading a file to cooperate is a basic requisite for achieving superior system performance. This alone, however, will not suffice in a P2P context, because the highly asymmetric nature of residential broadband links does not align well with the requirements of P2P content distribution. In particular, the downstream bandwidth of users is often an order of magnitude higher than their upstream capacity [25]. This suggests that a set of users that are all downloading from each other can never saturate their downstream links by default, as important upstream capacity is missing. Thus, a sophisticated incentive design needs to find a method to compensate for this lack of capacity. This is only possible if other users provide their uplink capacity while not downloading (i.e., they perform the role of an altruistic data source). Therefore, data sources not only ensure the availability of content, but also increase download performance, which highlights the necessity to incorporate these users into incentive design.

Challenge 3: *Ensuring Fairness in the Face of User Heterogeneity.*

Most of the Internet access links are not only highly asymmetric, but also vary significantly with regard to their performance capabilities [25]. The skew in user upstream

capacities is extreme; a small fraction of higher provisioned nodes (10%) hold more than 79% of the total capacity [50]. This observation, however, induces a serious performance-fairness dilemma. In particular, to achieve the best possible system throughput, it is necessary to keep higher provisioned nodes engaged as long as possible in the distribution process so as to sustain maximum aggregated system capacity. Indeed, it can be shown that average download throughput increases significantly, if high capacity users leave the system last [31].

From a user's perspective, however, the above situation seems unfair as system throughput is achieved by exploiting peers residing at the higher end of the capacity spectrum. Moreover, precisely because of the selfishness of users, systematic unfairness should be avoided by all means. This is because a peer's upload contribution is typically a tuneable parameter that can be freely configured by the participating users (potentially resulting in high capacity nodes withholding their resources). Consequently, systemic unfairness will likely result in performance collapse in the long-term. Therefore, a sophisticated incentive design needs to find a method to cope with this inherent performance-fairness dilemma.

1.4 RESEARCH GOALS

Motivated by the above issues, the *key objective* of this thesis is to strengthen and improve the download performance and file availability of P2P content distribution systems. This objective can be further decomposed into three core research goals:

- To investigate the shortcomings of current P2P content distribution systems with regard to providing continuous content distribution. Specifically, to understand *how* existing incentive mechanisms fail to fulfil performance and availability requirements.
- To design and build a new P2P incentive mechanism that *addresses* any discovered performance and availability limitations of existing systems.
- To show the superiority of the new incentive mechanism; specifically to *(i)* validate that improved *performance* and *availability* can be achieved compared to existing tit-for-tat mechanisms, and to *(ii)* ensure that this can take place in a *robust*, *fair*, and *low overhead* manner considering current operating environments.

1.5 THESIS ORGANISATION

This thesis is structured into seven chapters.

After this introduction, Chapter 2 *"Background and Related Work"* details basic principles and important terminology in the domain of P2P content distribution. Following this, an abstract overview of a set of prominent P2P protocols used for online content distribution is given. Subsequently, related work in the field of P2P incentive mechanisms is discussed.

Chapter 3 *"Measurements and Analysis of the BitTorrent System"* deals with two major measurement studies that we performed in the BitTorrent system. The purpose of these studies is to validate our critique of P2P content distribution with respect to providing continuous content distribution and to discover vital properties that cause these limitations.

Chapter 4 *"Analysis of Solution Space"* studies a set of intuitive solution approaches with regard to the issues found in Chapter 3, both to narrow the solution space and to shape a more sophisticated incentive design. A further purpose of this chapter is then to derive a set of key requirements that a solution must fulfil to overcome the performance and availability issues discovered.

Chapter 5 *"A New P2P Incentive Mechanism – FairSwarm.KOM"* focuses on the design of a novel P2P incentive mechanism that is devised to meet the solution requirements posed in Chapter 4. Within this chapter, a conceptual overview of the solution's basic concepts and major building blocks is given. Following this, each building block is presented and discussed in detail, while important design decisions are justified.

Chapter 6 *"Evaluation of FairSwarm.KOM"* evaluates the devised P2P incentive solution with a focus on performance, availability, fairness, robustness, and overhead. To this end, extensive trace-based simulations are performed, taking our BitTorrent measurement data as an input.

Chapter 7 *"Conclusion"* first summarises the content and main findings of this thesis. Following this, the main contributions of the thesis are highlighted and the research goals are revisited. Finally, the thesis is concluded by presenting areas of potential future work opened up by this work.

2 BACKGROUND AND RELATED WORK

This chapter introduces the basic principles and important terminology in the context of P2P content distribution. After this, we give an abstract overview of a set of prominent P2P protocols used for online content distribution. Finally, related work in the area of incentives for P2P systems is discussed, particularly in P2P online content distribution.

2.1 INTRODUCTION

The overarching goal of this chapter is to provide relevant background information to assist in the understanding of the following chapters. In particular, in Section 2.2, we describe the major components of a P2P content distribution system and explain their key functionality. Thereby, important terminology is introduced. Following this, in Section 2.3, we give an abstract overview of the most prevalent P2P systems used for online content distribution, namely Gnutella, eMule, and BitTorrent. The information presented form the basis required to understand the analysis performed in the remainder of the thesis. Finally, since a core research goal of this thesis is to design a new P2P incentive mechanism, Section 2.4 reviews related work in the field of P2P incentives. To this end, incentive mechanisms are classified into three categories: direct reciprocity (bilateral schemes), indirect reciprocity (multilateral schemes), and virtual currency systems. The chapter then concludes with a brief summary containing the main points to keep from this chapter.

2.2 BASIC PRINCIPLES OF P2P CONTENT DISTRIBUTION

Any form of P2P content distribution can be reduced to certain basic components that provide the required functionality to distribute content to a set of interested end-users. This section briefly delineates those components, including a brief discussion of the advantages and drawbacks of different approaches to realise either component.

2.2.1 *Overview*

Modern P2P content distribution systems usually split a file into smaller parts, which are named *chunks* in the following. This allows downloading peers to act as servers to other peers as soon as they have received a complete chunk, which is obviously quicker than waiting for a complete file. From the downloader's point of view, this method also allows for downloading a file from multiple sources at the same time.

In literature, the above technique is often referred to as *multi-source downloading*. As shown by [129, 11, 32], multi-source downloading has two major advantages compared to classical client/server approaches. First, it exploits the upstream capacity of the users and therefore increases the throughput of the system, which ultimately decreases download times. Second, the approach is inherently self-scaling while avoiding the limitations of client/server approaches, i.e., it continuously adapts system capacity because of increasing user populations and access connections.

As highlighted by [32], to best capitalise the upstream capacity of the users in the data distribution process, it is mandatory *(i)* to engage users as fast as possible, and *(ii)* to keep users engaged as long as possible.

Evidently, users can only be integrated into the distribution process if they are useful to others. Therefore, it is important to maintain a high degree of chunk diversity by

keeping the number of copies of each chunk approximately the same. This prevents peers from having to wait for extended periods for rare chunks that only a few other peers possess.

The architecture of modern P2P content distribution systems basically consists of two architectural elements: content *discovery* and content *delivery* [4, 73]. The functionality of each of these elements can be briefly summarised as follows:

1. *Content Discovery*: The objective of the content discovery component is to find *source* peers in the system, capable of serving the file. In multi-source downloading, a potential data source is a peer that possesses at least one file chunk. In this regard, we distinguish between source peers that are already in possession of the entire file, so-called *seeders*, and sources that possess only a subset of the file, named *leechers*.

2. *Content Delivery*: The delivery component defines a 'rule set' how chunks are efficiently forwarded from discovered data sources to interested receivers. To this end, important decisions are made within this component. First, a user must decide how many peers to serve simultaneously. Indeed, each peer's *outdegree* k varies depending on the chosen peer organisation strategy that can range from a linear chain to a tree or mesh [11]. Further, with respect to the chosen distribution model, a peer must also select the users it wants to serve data to and subsequently decide which chunk to download next. All three, the distribution model, the peer selection strategy, and the chunk selection strategy have a major impact on download performance [129, 11].

In essence, the delivery of content in a cooperative content distribution system consists of two steps. In step one, potential data sources for a certain file are discovered. In step two, the discovered data sources are interconnected with interested receivers simultaneously demanding the file. Subsequently, the actual content delivery process starts following a certain delivery technique.

We next provide a brief overview of techniques to realise source discovery, and briefly discuss the advantages and drawbacks of different peer organisation strategies.

2.2.2 *Discovery Mechanisms*

In order to discover source peers in P2P content distribution systems, either a centralised or decentralised approach can be taken [27]. In addition to this, these two approaches can also be combined together to form a hybrid source discovery model [23, 24]. We now present an abstract overview of the two basic discovery models. For more details, we refer the interested reader to [116].

Centralised Source Discovery

For system designers, centralised source discovery is often a popular choice, largely because of the ease of implementation and increased control of the system. In such an approach, a central entity acts as entry point to the system at which every participant must register. This component periodically receives reports from all connected peers, e.g., to build up a search index of all available files and data sources [27]. Through this global view, system designers can easily implement different functionalities to fulfil specific needs. For instance, as exemplified by the Napster[1] system, the central server could answer keyword-based queries, initiated by participating users. Alternatively, the central server could be configured to always return data sources that are in direct proximity of the requesting nodes, e.g., reside in the same Internet Service Provider or

1 http://www.napster.com

country. This is highly useful to achieve traffic locality and to improve the download performance of the users [35, 126, 1, 99].

The major disadvantage of centralised source discovery is robustness and scalability. For one, the entire system relies on the proper functioning of the central entity. Consequently, the sudden failure of it often induces a system breakdown. Furthermore, apart from being a single point of failure, a central component to lookup sources can easily be the subject of malicious nodes trying to disturb or even destroy the entire system. Finally, even powerful centralised lookup entities (e.g., consisting of multiple servers) are limited in their CPU power and bandwidth capabilities. For instance, according to [12], even the high performance BitTorrent tracker BitComet only supports 80,000 torrents and 800,000 users. In practise, however, P2P content distribution systems can easily exceed this threshold by an order of magnitude in terms of user population size (see Chapter 3). A common remedy to improve robustness and scalability is clearly to use multiple trackers at the same time.

Decentralised Source Discovery

Decentralised source discovery is usually implemented in a P2P overlay network, which provides certain search capabilities, ranging from identifier lookups to location-based and full-text searches [63, 64, 65, 116]. The overarching goal of decentralisation is to foster scalability and provide increased robustness, e.g., the approach still works regardless of whether some peers fail in the overlay network. Decentralised source discovery can be implemented with the aid of unstructured or structured overlay networks[2]:

- *Unstructured Overlays.* In unstructured overlays, each peer maintains a number of overlay connections to other peers in the system, referred to as its local neighbourhood. Those distinct connections form the only way to send or receive overlay messages from other nodes. To lookup data sources, peers can either flood request messages in their local neighbourhood (e.g., the Gnutella system [20]) or use random walks by sending messages to a subset of their neighbours who in turn repeat this process [18]. Intuitively, these two generic approaches can also be combined, as exemplified by the BubbleStorm system [120]. It is worth noting that unstructured overlay networks usually face the problem of non-deterministic results; a peer initiating a search may or may not receive responses from other nodes in the system. That is, there is no abortion criteria to find out whether the search is complete, i.e., if all nodes possessing the desired content are reached.

- *Structured Overlays.* Structured overlays provide a unique mapping between data objects and peers responsible of storing these. These addressing schemes are typically implemented by so-called *distributed hash tables* (DHTs) [104, 117, 83, 106, 135]. As an enhancement of a normal hash table, a DHT provides a distributed lookup/store functionality for (key, value) pairs. To achieve this, participants and data items are assigned identifiers from a sufficiently large identifier space. The identifier (ID) of a data item is usually its hash value. Peers either choose a random identifier or derive it by calculating a hash value of their own IP address, for instance. To obtain the distributed buckets, peers are only responsible for storing data items with IDs that are close to their own ID. This notion of closeness usually varies between the different DHTs [4, 56]. To find these nodes in the network, peers perform a prefix matching between their overlay contacts and the ID of the object to be looked up. Depending on the metric applied, the 'closest' node is then chosen to receive the query, which repeats this process until the target node is found.

2 In this context, the term structure refers to a peer being assigned for a specific task and/or position in the network.

A major advantage of DHTs is that ID lookups are deterministic; they always find the data object belonging to a certain identifier, if existent. On the other hand, compared to unstructured overlays, it must be noted that, in DHTs, the realisation of keyword-based searches is much more complicated. For instance, due to the prefix-based routing, there is no direct way to support wildcard searches for queries consisting of multiple words. In fact, as exemplified by the eMule system, one technique to overcome this is to hash keywords and compute an intersection on the obtained results [66, 114]. This involves, however, several DHT lookups per query, and, consequently, there is a clear trade-off between lookup performance and communication costs.

2.2.3 *Delivery Mechanisms*

To replicate chunks from source peer(s) to interested receivers, system designers can chose from three different architectures: a linear chain, tree, and mesh [11]. In the following, we briefly discuss the properties and applicability of these approaches in the context of online content distribution.

Importantly, the coordination of the content delivery process must rely on the collaboration of individual users, without having to resort to any centralised control. Additionally, the sudden failure or departure of a node, referred to as *churn* (or synonymously peer turnover), is likely to occur and must be taken into account. Finally, recall that most of the access links are highly asymmetric such that users can often download content faster than they can upload it [25].

Linear Delivery

The simplest and most straightforward approach is to organise peers in a linear chain (cf. Figure 2). In this *linear* architecture, every peer serves the entire file to another peer and then disconnects. This means a peer's outdegree is one ($k = 1$). Nevertheless, a peer can start serving as soon as it owes one chunk. Peer selection is rather static as users have only one ingoing and one outgoing connection. Also, there is no chunk selection mechanism, since file fragments are sequently forwarded from one node to another.

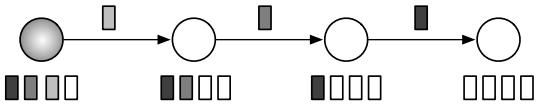


Figure 2: Illustration of a linear delivery architecture.

A major advantage of a linear delivery architecture is that peers do not have to maintain state about which chunk has already been downloaded by their neighbours. Instead, every downloaded chunk is useful for the peers' downstream neighbours. Further, due to the simpleness of the approach, it is straightforward to move the higher provisioned nodes towards the seeder. This optimises distribution speed as the peer with the slowest upstream capacity is bottleneck, i.e., it determines the delivery rate of the downstream nodes.

This simplicity and low overhead, however, comes at the expense of some attributes that prove themselves to be disadvantageous for P2P content distribution. First, a linear chain inherently lacks robustness against churn; a sudden crash or disconnect of one node may disconnect all downstream peers in the delivery chain. Second, peers at the end of the chain will experience a long idle period until the actual download starts. Lastly, since the outdegree k is constantly one, the users' downstream capacity will only be poorly utilised because of the link asymmetry of current Internet access connections.

Tree-based Delivery

In a k-*tree* architecture, the seeder is root of a distribution tree and uploads to k peers in parallel (cf. Figure 3). As soon as an 'inner'-node (non-leaf peer) has downloaded one chunk, it starts to forward it to k child nodes in parallel. Leaf-nodes only obtain data, but do not upload it to others. Similar to a linear chain, peer selection is rather static as the tree structure remains unchanged until all peers have completed their downloads. Also, chunks are sequently forwarded from parents to leaf nodes.

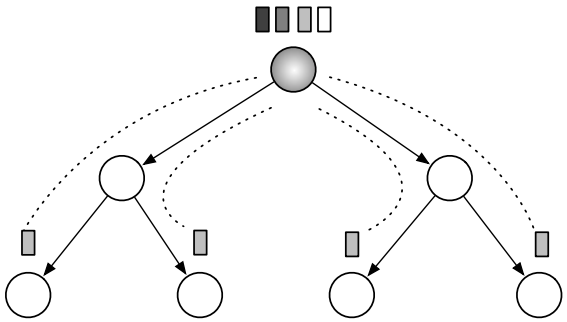


Figure 3: Illustration of a tree$^{k=2}$ architecture.

When compared to a linear chain, the k-*tree* architecture offers some important improvements. First, the approach is much more robust against peer turnover. In particular, depending on the tree position, the crash of an inner-node affects significantly less downstream users. Moreover, the sudden failure of a leaf-peer (half of the users) does not impair the downloading process at all. Finally, the waiting-time for content is significantly lower, as the initial seed can inject data chunks to k nodes in parallel, which immediately repeat this process.

In the context of P2P content distribution, however, care must be taken when choosing this approach as delivery architecture. In general, each serving node has one incoming and k outgoing connections. This suggests that a k-*tree* performs best if peers can upload k-times faster than they download. However, this is exactly the opposite to what today's Internet connections offer. Further, when finishing a download, each non-leaf peer has uploaded the file k times, whereas the upstream capacity of the leaf nodes remains unused.

A more sophisticated approach is to arrange nodes in a parallel k-*Tree*, as proposed by SplitStream [17]. Compared to a standard k-*Tree*, this architecture strives to also leverage the upload capacity of the leaf nodes. To do so, peers are organised into k different trees such that each user is a non-leaf node in at most one tree, and a leaf node in the remaining $k-1$ trees (cf. Figure 4). Subsequently, each chunk is further subdivided into k blocks and the k-th block is served into its k-th tree, respectively.

Compared to a standard k-*Tree*, a parallel k-*tree* is less sensitive to churn and additionally leverages the upstream capacity of the leaf-nodes. Further, each peer downloads k parts of the file from k peers in parallel, while it, at the same time, serves data to k other peers. Therefore, each peer receives exactly what it uploads. Ignoring the complexity of maintaining such a tree without a central coordinator, the parallel k-*tree* assumes that access connections are symmetric, still making the approach ill-suited for online content distribution over the Internet.

Mesh-based Delivery

Organising peers in a linear chain or in a tree-based architecture results in a static topology that is complex to maintain without centralised coordination (e.g., in the case of sudden failures). In contrast to this, mesh-based architectures are based on the premise that every peer can interact with *any* other node in the system (cf. Figure 5).

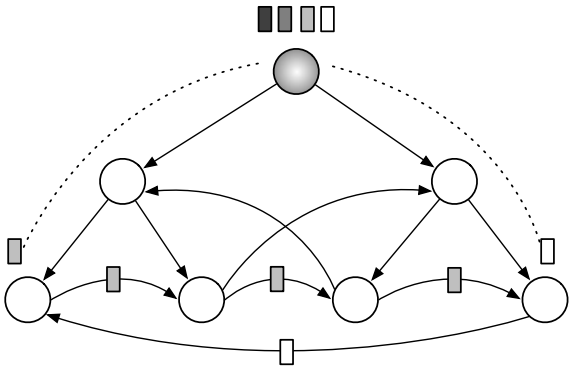


Figure 4: Illustration of a parallel tree$^{k=2}$ architecture.

Also, there is no fixed chunk selection strategy and peers are allowed to change their peering partners at will at any time without notice, i.e., they are not limited to a fix overlay structure with fix peer sets.

The topology of such mesh-based system is, thus, continuously changing over time and is only determined by two factors: *(i)* the maximum amount of neighbours a peer wants to simultaneously upload to (the outdegree k) and *(ii)* the chosen peer selection strategy defining whom to upload next. The peer's outdegree is typically a user-specific setting and computed over the node's available upload bandwidth. The peer selection strategy, on the other hand, can be implemented in different manners. For instance, peers may prefer nodes with high upload capabilities to accelerate the distribution of rare chunks in the network. Alternatively, they may also favour particularly cooperative nodes in the system.

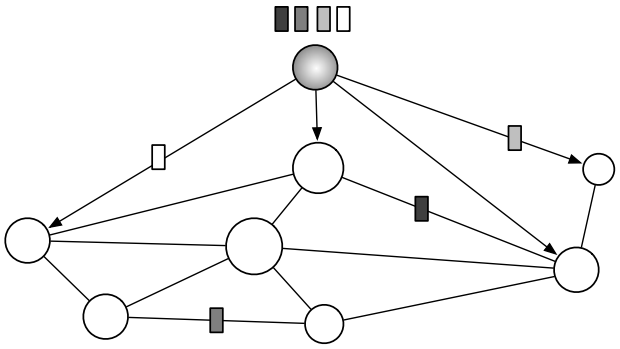


Figure 5: Mesh architecture with varying peer outdegrees.

Furthermore, this flexibility with regard to peer selection allows mesh-based delivery architectures to quickly adjust to sudden failures, making them highly robust to churn [45, 44]. Moreover, it has been demonstrated that in mesh-based architectures, the locally-rarest-random strategy in requesting missing chunks is highly efficient [62]. In particular, it can be shown that such architectures replicate chunks in the system with an exponential capacity of service, thereby clearly outperforming linear chains or tree-based architectures in terms of download completion times [129]. On the other hand, to enable mesh-based systems based on sophisticated chunk requesting strategies, peers need to exchange control traffic which clearly induces additional communication overhead and complexity.

2.3 WIDELY DEPLOYED P2P CONTENT DISTRIBUTION SYSTEMS

Upon presenting the relevant details to gain a basic understanding about important key principles of P2P content distribution, we next provide an abstract overview of a set of prominent P2P protocols used for online content distribution. Our discussion focuses, however, only on those systems that are currently most relevant, i.e., widely used in terms of user population and proportion of the Internet share. According to this metric, the different systems are presented in a chronological order.

2.3.1 *Gnutella*

The Gnutella protocol was developed in 2000 and is one of the first P2P content sharing systems [20, 28]. Due to its early development state, however, the system now plays only a secondary role in online content distribution [49]. It belongs to the so-called *pure* P2P applications that do not rely on any centralised component [27]. Instead, all participants are interconnected in an unstructured overlay network. To lookup content, peers either flood queries to all overlay neighbours (version 0.4) or rely on super peers to find potential data sources (version 0.6) [61]. As such, the Gnutella protocol implements a decentralised source discovery mechanism.

The delivery mechanism of Gnutella is very simplistic. In particular, in Gnutella 0.4, data transactions always relate to entire files, instead of small file chunks. Thus, peers can only download from a single data source, and, consequently, there is no support for multi-source downloading. Gnutella 0.6 attempts to improve this by supporting HTTP range queries for different file fragments. However, to use multiple download sources, it still leaves the design of the delivery process up to the clients involved.

As one of the first P2P content distribution systems, Gnutella does not implement any contribution incentives and therefore solely relies on the altruism of the nodes to contribute their resources. This assumption, however, has severe consequences for system performance; due to the lack of incentives, the Gnutella network is largely plagued by free-riding during recent years, i.e., users tried to download content from others while not contributing themselves [2, 48]. As a result, 15% of the participating users in Gnutella provided 98% of the content, clearly demonstrating the systematic unfairness in the process [48]. Because of this important lesson, subsequent systems therefore explicitly built contribution incentives into their design.

2.3.2 *eDonkey/eMule*

The eDonkey [53] protocol is an open file sharing protocol that was initially developed in 2000 by MetaMachine Inc. In 2005, however, this company officially discontinued their development by following a 'cease and desist' letter from the Recording Industry Association of America (RIAA). Nevertheless, the eDonkey network is still available through the popular file sharing application *eMule*, which has taken over the further development of this protocol [66]. According to the iPoque study [49], the eMule network is currently the second most popular P2P file sharing application in many regions of the world. As shown in Table 1, in 2008, it accounted for 24% of the total P2P traffic in Germany, even though this number has been continuously decreasing during the last years.

The eMule file sharing network consists of hundreds of eMule servers and hundreds of thousands of eMule clients [66]. eMule servers are solely responsible for indexing files, and for distributing IP addresses of other eMule servers and clients [66, 79]. As such, they do not store any files, but act as a communication hubs to connect eMule clients together. Each eMule client, on the other hand, is pre-configured with a list of eMule servers and a list of files it wants to share in the system. To enter the network, a client may connect to multiple servers at the same time to retrieve

Protocol	2007	2008	Change
BitTorrent	66.70%	70.77%	+6.11 %
eDonkey	28.59%	24.22%	-15.27%
Gnutella	3.72%	1.75%	-53.06 %
Other	0.99%	3.77%	+280.54%

Table 1: The proportion of the most popular content distribution systems out of the total P2P traffic in Germany [49].

information about desired files and available data sources. Upon bootstrapping, each eMule client maintains several hundred TCP connections to other clients for uploading and downloading content. Since most of the eMule servers have been overloaded in recent years, the eMule network has additionally resorted to a decentralised resource discovery scheme, which utilises the resources of the individual peers. In particular, it makes use of the Kademlia distributed hash table [83], which is also known as the Kad network [115].

To enable data exchange between peers, eMule splits files into individual fragments, which are called *parts*. Peers can share individual parts before they have downloaded the entire file, thereby enabling a mesh-based multi-source downloading. Unfortunately, the part size is hard-coded into the protocol with a rather large value of 9.28 megabytes [66]. Individual parts can be verified using the hashes from a hash list, and, correspondingly, files are identified in the network using their unique hash identifiers. To select which part to download next, eMule clients prefer parts that are rarest in their local neighbourhood (the locally-rarest-random requesting strategy).

To tackle free-riding, eMule employs a credit-based *tit-for-tat* incentive strategy that rewards the long-term cooperativeness of users. For this purpose, peers possess permanent identifiers that can be used to identify nodes across time and multiple download sessions. To keep track of user contributions, however, peers solely use local histories and direct observations. That is, they do not exchange information about third-party experiences. Peer selection is then performed by preferring nodes that exhibit particularly high share ratios, defined by the amount of data provided and consumed over time.

2.3.3 *BitTorrent*

The BitTorrent protocol was developed by Bram Cohen in 2001 [21], and is nowadays the de-facto standard for scalable P2P content distribution. Alone in Germany, BitTorrent accounts for more than 70% of the total P2P traffic (cf. Table 1), which is more than twice the total HTTP traffic [49].

The BitTorrent 'ecosystem' consists of peers, trackers, and torrent discovery sites (cf. Figure 6) [133]. As opposed to all previous P2P content distribution systems such as Gnutella or eDonkey/eMule that organise peers sharing *multiple* files together into an overlay network, BitTorrent organises peers interested in the *same* file into a so-called BitTorrent swarm (or synonymously called a torrent) [43].

Each single BitTorrent swarm is identified by a unique identifier, called the *infohash*, which is typically stored in the `.torrent` meta information file. To join a particular swarm, a peer must first learn about its existence from a torrent discovery site such as Mininova, PirateBay and IsoHunt. There are dozens of discovery sites, each representing a distinct BitTorrent community. Table 2 shows the most popular indexing sites, obtained from the Web-traffic monitoring site Alexa[3]. Each torrent discovery site

3 http://www.alexa.com

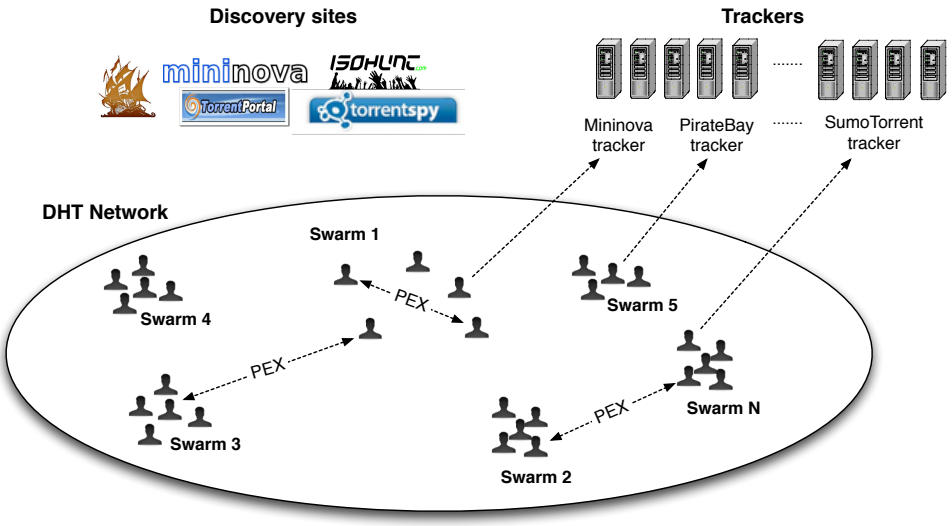


Figure 6: The BitTorrent ecosystem [133].

Discovery Site	Alexa Rank	Location
Mininova.org	88	Netherlands
Thepiratebay.org	109	Sweden
IsoHunt.com	219	Canada
Torrentz.com	225	Netherlands
Btjunkie.org	454	Germany

Table 2: Top-5 most popular torrent discovery sites.

maintains a database of `.torrent` files that contains the torrent name, the hashes of all file chunks, the IP-address of the responsible tracker, and the infohash.

After discovering the file of interest on one of these sites, a peer downloads the corresponding metafile (i.e., the `.torrent` file). Subsequently, it contacts the tracker by sending the appropriate infohash, which returns a random subset of the peers (typically 50 nodes) currently participating in the swarm. The new node then tries to establish a TCP connection to these nodes in order to be incorporated into the current download process. To support peer heterogeneity, the maximum number of peers each node connects to is limited by the node's uplink capacity. Therefore, the *neighbour set* size of each node is variable in BitTorrent.

To balance the tracker load and provide increased robustness, peers can further learn about other peers in the swarm using distributed hash tables (DHTs) and peer gossiping. For instance, peers using the Azureus client interconnect to a Kademlia-based DHT that takes the torrent's infohash as a key [30]. Looking up a specific infohash in this DHT returns a list of other Azureus peers currently participating in the swarm. In addition to this, uTorrent and Azureus clients have defined their own protocol specification, the so-called the Peer Exchange protocol (PEX) [103], allowing peers to mutually exchange their neighbourhood lists. Note that both peer discovery mechanism (i.e., DHT and PEX) return, however, only a subset of the information that is available from the tracker. As such, BitTorrent implements a hybrid source discovery mechanism.

A file is split into chunks that are typically between 256 KB and 4 MB in size, which varies depending on the size of the file. While BitTorrent peers maintain TCP connections to roughly 50-120 neighbours, they solely upload to, or *unchoke*, a small

number of them. Each user may individually adjust this number of unchoke slots (its outdegree k), which is by default 5 [21]. The peer selection strategy is termed the *choking algorithm* in BitTorrent, responsible for deciding whom to upload data to. To this end, BitTorrent implements a real-time *tit-for-tat* incentive policy to favour peers that have recently provided the highest download rates. Similarly to eMule, for the decision making (i.e., downloader selection), peers solely use local contribution histories and direct experiences with other nodes.

To guide the peer selection process, users inform one another of the chunks they have downloaded. That is, upon connection establishment, peers mutually exchange a bit array of their chunks, called a *bitfield*. As soon as a peer successfully receives and verifies a new chunk, the bitfield is updated by sending per-chunk *have* messages. The common strategy to decide which chunk to download next is *rarest-first*. In this manner, peers specifically favour chunks with the least number of copies, computed across all neighbours' bitfields. As such, the BitTorrent protocol is an archetype for mesh-based content delivery.

2.4 RELATED WORK

The focus of this work is to design a superior incentive mechanism that strengthens and improves the download performance and file availability of (existing) P2P content distribution systems (cf. Section 1.4). In order to establish a basis for comparison and discussion, this section presents related work based on existing incentive strategies in P2P content distribution. Since much work has gone into encouraging users to contribute their resources in these systems, we first present our methodology to classify those works. Subsequently, we give an overview of the most recent approaches of each category and emphasise important findings that are relevant for the remainder of this thesis.

2.4.1 *Overview*

So far it has been shown that P2P content distribution systems consist of a content discovery and content delivery scheme. The former is necessary to discover potential data sources, while the latter defines how these sources are interconnected to quickly replicate content. In early system designs (e.g., Gnutella), it was implicitly assumed that peers are behaving collaboratively and are always willing to contribute their upstream resources for the benefit of the system. However, in practise, it can be observed that many users tend to strategise, i.e., they do not contribute resources unless they receive a benefit for it [2, 48]. Importantly, this selfishness of users can be observed in most of the popular P2P content distribution systems, including eMule and BitTorrent [132].

Therefore, since a successful content delivery mechanism has to rely on peer participation, system designers need to find methods to motivate users to cooperate with the system. To achieve this, *incentive mechanisms* strive to provide a profitable bonus to users for serving content to other participants in the system. Within the context of content delivery, such a bonus may be that cooperative peers achieve higher download rates for content they desire, either immediately or in future downloads.

The download rate of a user is usually limited by the sum of upstream capacity that other peers are willing to provide. To promote competition among users and thus to incite cooperation, it is therefore necessary for peers to keep record of the contributions of other users. Additionally, to increase the efficiency of this, peers may also exchange messages with third-parties to inform other users about the cooperativeness of their interaction partners. There are different approaches to spread this information in the system. They all differ in information quality, propagation speed, traceability, communication overhead, and other criteria. In this work, we order those approaches by the knowledge base a peer obtains in the network. This ranges from approaches

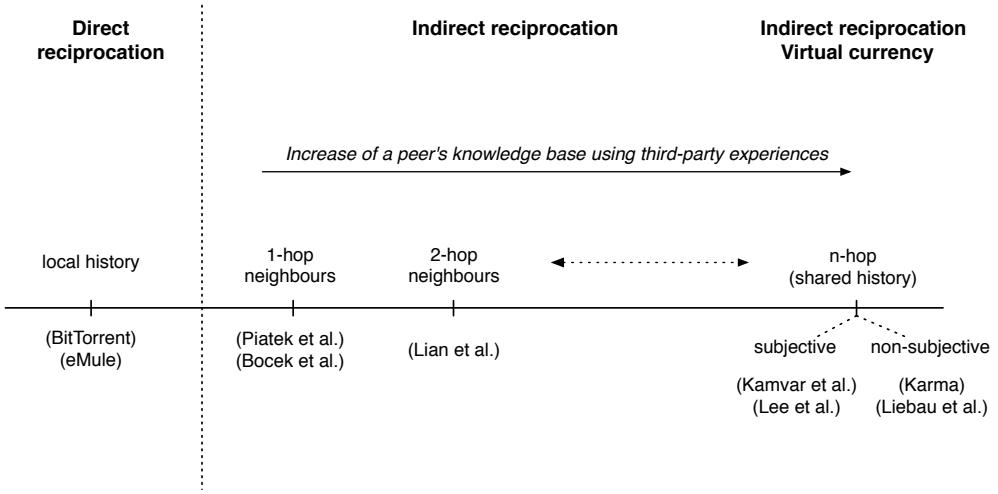


Figure 7: Overview of categories of incentive mechanisms and examples.

considering only local experiences to approaches where all peers share and have access to the same information base, a so-called *shared history* (see Figure 7).

A term that is useful when talking about incentive mechanisms is *reciprocity*. In general, reciprocity means rewarding a peer for its contribution. To this end, we distinguish between *direct* reciprocity (bilateral incentives), i.e., servicing a peer with content because it has serviced us before, and *indirect* reciprocity (multilateral incentives), i.e., servicing a peer with content because it has serviced some other peer(s) before. In addition to this, some works propose a *virtual currency* in which users can only obtain services from the system that do not exceed their account balance. We next present related work based on these three categories.

2.4.2 *Direct Reciprocity*

Incentives based on direct reciprocity only reward those nodes with which a user has had a good experience in the past. In particular, each user maintains a *local* history of the contributions of other nodes, which is then used to rank requesting peers later on. There are two different criteria that can be used to rate peers, i.e., based on their provided download rates, or, alternatively, based on their ratio of data provided and consumed.

To the best of the author's knowledge the only (deployed) system that implements a *rate-based* incentive mechanism is the BitTorrent protocol [21]. More precisely, it employs a so-called rate-based tit-for-tat strategy that periodically (every 10 seconds) ranks users according to their provided download rates, measured during the last 20 seconds. Using this ranking, the top k peers in this list (default $n = 4$) are unchoked and their requests accepted. The effectiveness of BitTorrent's incentive mechanism has been studied in detail, e.g., [51, 100, 10, 71, 31]. By using a five months long tracker trace log of a large torrent, Izal et al. [51] reported that BitTorrent is highly effective at sustaining flash crowds, and, due to the instantness of BitTorrent's tit-for-tat servicing policy, clients observed extremely high download rates. Qiu et al. [100] propose a simple fluid model to capture BitTorrent's performance. Amongst other things, they find that the system scales fairly well and that the delivery process is very effective, mainly because of BitTorrent's incentives. Bharambe et al. [10] confirm this and show through large-scale simulations that BitTorrent is remarkably robust at ensuring high uplink bandwidth utilisation. Legout et al. [71] use testbed analysis to demonstrate that BitTorrent's incentive mechanism forms download clusters that are proportional to the peers' upload contribution. Finally, Bin et al. [31] demonstrate that there is an

fundamental trade-off between download performance and fairness in BitTorrent, and that BitTorrent's current implementation is only one point in the design space.

In contrast to all these results, there are also a few studies that have challenged the robustness, fairness, and performance properties of BitTorrent's incentive mechanism. For instance, [77, 78, 112, 93] expose situations in which BitTorrent peers (i.e., particularly seeders) can be systematically exploited by free-riders, and, therefore, proposed several refinements. Further, in contrast to the work of Legout et al. [71], Piatek et al. [93] find that rate-based incentives are inherently unfair in terms of upload contribution and download reward. This observation is also confirmed by Levin et al. [72]. In addition to this, the measurement analysis of [96, 94] reveal that, in BitTorrent, user download performance is sometimes quite low and even a 100-fold increase in upload contribution only yields a very slight performance improvement.

It is commonly known that BitTorrent's rate-based incentive mechanism only applies in the context of a single swarm, and thus does not incite users that have completed their downloads (seeders) to remain in the system [72, 94, 10]. Recent works such as [43, 42, 10, 84, 113, 122] demonstrate, however, that seeders have an important impact on download performance and availability in BitTorrent. Specifically, Guo et al. [42] propose a fluid model of the lifespan of torrents in BitTorrent to find that most torrents are short-lived. To tackle this, Yang et al. [130] propose a variation of BitTorrent's rate-based incentive mechanism that allows to recognise contributions from users in other torrents, enabling what we call *cross-torrent bartering* (cf. Section 4.2). This approach, thus, inherently provides incentives for seeding.

In contrary to rate-based incentives, pairwise *volume*-based approaches, on the other hand, keep track of the users' amount of data provided and consumed. For instance, the eMule/eDonkey system uses this approach to establish a credit system that rewards the long-term cooperativeness of the clients with reduced download times [66]. This incentive approach operates across time and file downloads, allowing for what we call *cross-torrent tit-for-tat* (cf. Section 4.2). To the best of the author's knowledge the only existing performance analysis of this incentive mechanism is that of Pucha et al. [98]. To this end, the authors find that the majority of peers in eMule (90%) download with rates below 10 KBps, which is extremely poor when considering the users' downstream capacities [25]. Apart from the eMule network, in the Swift system [118], peers define a total credit for all users that they interact with. Download requests are only satisfied if the credit value is greater or equal to the requested data. In Sherman et al. [109], each user maintains a deficit counter for each interaction partner that is computed by the difference between data send and received. In contrast to the eMule system, however, the deficit counter is valid for only *one* file download allowing to quantify the instant cooperativeness of each user. Interaction partners with the lowest deficit are then chosen for uploading. Finally, in [54], Jun et al. propose a byte-level tit-for-tat incentive mechanism to tackle free-riding in the BitTorrent system, which is similar to pairwise volume-based incentives.

It is worth noting that all currently deployed P2P content distribution systems solely implement incentive strategies that are based on direct reciprocity. For instance, BitTorrent employs an instant tit-for-tat incentive, while eMule opts for a volume-based long-term tit-for-tat incentive strategy. This design choice is intentional because the major advantage of any *bilateral* incentive strategy is that decision making is solely based on direct observations and local histories. This makes those approaches highly robust to strategic/malicious attacks such as peer collusion [33, 74] or Sybil attacks [26]. The drawback of direct reciprocity is, however, that symmetric interests as well as repeat interactions between peers are required. While these conditions can be easily fulfilled for the download of a single file object (e.g., when applying multi-source downloading), it is unclear how these approaches perform across multiple file downloads. In particular, modern P2P content distribution systems consist of millions of online users, and thus

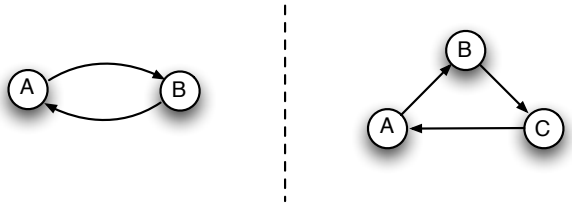


Figure 8: Comparison between direct and indirect reciprocation.

repeat meetings between two peers might be infrequent. We will thoroughly investigate this issue in Chapter 4.

2.4.3 *Indirect Reciprocity*

In contrast to direct reciprocation, *indirect reciprocity* relies on direct observations, and additionally on information from third-parties that are distributed among the nodes in the system. This means, instead of only considering peer B's direct contribution to peer A, peer A may exchange information with an overlay neighbour C, allowing it to also see B's contributions to C (see Figure 8). This knowledge base can be arbitrarily extended when peer A also shares information with its neighbours' neighbours and so on. Pushing this direction to its extreme arrives at a *shared history* in which all peers share their local observations among each other, and the contribution of a node becomes *globally* visible.

While direct reciprocity assumes a symmetry of interest between the peers, indirect reciprocation is also able to cope with asymmetric demand and, thus, does not necessarily require peers to repeatedly meet each other. This greatly improves the coverage of known peers in the network; a peer may judge another peer even if it has not interacted with it before, simply by using another peer's experience. However, this advantage comes at the expense of other properties. For instance, the scale of information produces network traffic that may exponentially increase with each additional level of indirection [131]. In addition to this, trusting other peers makes an algorithm vulnerable to issues such as Sybil attacks, collusion, or whitewashing [34, 90, 74]. Therefore, when designing *multilateral* incentive strategies, all of these issues must be carefully considered and addressed, introducing a far greater complexity for incentive design.

During the last years, there has been a rich body of work on indirect reciprocation schemes, e.g., [22, 55, 16, 127, 69, 43, 94, 15, 87, 80]. Most of them have focused on several issues and also differ in the level of neighbours (or synonymously level of indirection) they use to compute *reputation* or *trust* values.

Lee et al. [69] propose a platform that can be used to implement cooperative applications on top of the Internet. In this system, peers assess the trustworthiness of other users by initiating a cookie search in the system. Cookies are assigned values on the [0,1] interval and evaluate a single transaction. Once the search is terminated, peers use transitive trust chains to estimate a global reputation value among the received cookies from other nodes. Guo et al. [43] sketch a very abstract mechanism to enable indirect reciprocation for the BitTorrent system. In this approach, the trackers in BitTorrent are responsible for detecting contribution cycles among the nodes across overlay swarms. If any exists, it is in the responsibility of the tracker to mediate data transfers along this cycle. The tracker is also responsible for punishing fraudulent peer behaviour (e.g., if a peer sends invalid data). Martinovic et al. [80] use mutually signed transaction receipts, so-called tokens, in order to protect P2P systems against file poisoning and free-riding. In contrast to the coupon-based approach devised in Section 5.3, tokens are created for every single transaction (i.e., the transfer of a file) and are thus unilateral. Furthermore, the authors use a flooding-based approach in order to make information accessible to

as many users as possible. Kamvar et al. [55] also focus on the problem of detecting peers providing inauthentic files in P2P systems. To achieve this, every participant is assigned a set of score managers, responsible for maintaining and computing a global trust value. To obtain this value, the managers recursively request the private histories of all the participants in the system. These local histories are used to compute a global trust matrix, similar to the PageRank [92] algorithm used by Google. Yu et al. [131] extend this work to show that the opinion of only a small minority is enough to have a good estimation of a peer's global reputation.

The works of [15, 94] go even further and promote the assumption that incorporating the information of one-hop neighbours is often sufficient to enable indirect reciprocation. In particular, Piatek et al. [94] show that the BitTorrent system is characterised by a small minority of peers that are extensively connected to other nodes in the network. This observation is exploited to enable indirect reciprocation between two less popular peers, sharing one of these highly popular nodes as common interaction partner. Similar to this, Bocek et al. [15] search for one-hop transitive paths between two peers, once the credit limit of a requesting peer is exceeded. Both works share the assumption that the common interaction partners are always *online* and are thus able to provide attestation receipts for prior interactions. However, as discussed in Chapter 5, across time *and* file downloads, this assumption holds for only 29% of the users. Therefore, in contrast to these works and to provide long-term incentives for seeding, the solution presented in Chapter 5 is designed to operate without the need for requesting intermediaries to attest contributions.

2.4.4 *Virtual Currency*

In addition to locally decided mechanisms or those based on indirect reciprocation, there is also a class of approaches that relies on a *virtual currency*, inspired by real life monetary payment. Peers in such schemes possess a predefined amount of virtual currency (or tokens) that they can use to request services, i.e., to download content. Upon receiving the requested content, the currency for this service provisioning is transferred to the servicing peer, who itself can then spend this money to receive content from another peer. Compared to reciprocity-based approaches, the account balance of a user is globally visible and contributions are not governed by the peers' subjective opinions. Also, peers have to continuously provide services to others, in order to not run out of money, preventing them from downloading new content. In addition to this, each peer — irrespective if it is a newcomer or long-time known — is worth interacting with if it has enough money to pay. Thus, there is no differentiation between the trustworthiness of users.

In literature, virtual currency systems can be classified into centralised and decentralised approaches. Centralised systems such as [113, 128] typically use a global instance, which is comparable to a bank, that records all transactions between peers. Each peer has a permanent identity in the system, which is connected to an account in the bank. This, of course, requires the bank to be trustworthy because all peers base the granting of transactions on the information provided by this bank. Consequently, every transaction that a peer performs becomes globally visible in the entire system. The major drawback of this approach is that each transaction has to be reported to a central instance, which results in high overhead traffic and limits scalability. For instance, the work of [113] requires at least one dedicated server to manage the downloads of only 200 clients.

Decentralised approaches typically distribute the functionality of a bank over a set of randomly chosen nodes in the system. MojoNation [88] is an example of a file sharing system that relies on a decentralised managed virtual currency, called "Mojo". In this system, each peer locally maintains a relative credit balance for each interaction partner. If the balance drifts too far in either direction, the debitor has to pay a "Mojo token" to

its partner, which can be used to request additional content from other nodes. Landa et al. [68] implements a distributed exchange economy based on social capital that allows peers to contribute resources to one set of nodes and use this contribution to obtain services from a different set of nodes. To achieve this, the authors propose a distributed accounting system that keeps track of the peers' contribution. BitStore [102] extends the BitTorrent protocol through a storage network that allows peers to offer their disk space for storing replicas of file chunks. This approach tries to raise the availability of torrents that have too few or no seeders. As reward for storing and serving those chunks, peers receive credits that they can use when the situation is reversed. The 'price' for uploading or downloading blocks is not constant, but found out by a market mechanism.

Other works [123, 39, 134, 76] provide decentralised micropayment schemes by relying on distributed hash tables. DHTs are, however, a particularly difficult venue for virtual currency as peers are transient and unreliable, and have no incentive to reliably maintain accounts for other users. To address this, these systems often assume *(i)* an already existing reputation infrastructure to rate the trustworthiness of the participants [76]; *(ii)* additionally replicate the functionality of a bank to multiple DHT nodes simultaneously (i.e., to provide increased reliability) [123]; *(iii)* and/or use threshold cryptography as countermeasure against strategic manipulation [134, 76].

2.5 CONCLUSIONS

This chapter has provided important information to assist in the understanding of the remainder of this thesis.

Section 2.2 discussed basic principles and important terminology in the field of P2P content distribution. Through this, it was shown that the architecture of such systems basically consists of two principle elements: a content discovery and a content delivery component. The former component is responsible to discover potential data sources for a certain file. The latter component, on the other hand, interconnects these sources with interested receivers simultaneously demanding the file. In this regard it was revealed that in current operating environments, mesh-based delivery architectures are best suited to replicate data chunks from source peer(s) to interested receivers. As opposed to a linear chain and tree-based approaches, these topologies can quickly adjust to user heterogeneity and peer turnover, and are only defined by the *(i)* peers' outdegree k and their *(ii)* neighbourhood size. More importantly, their flexibility in peer selection can be utilised to enable discriminative uploading — the basis for any incentive mechanism.

Section 2.3 provided an abstract overview of a set of prominent P2P protocols used for online content distribution. To this end, it was stressed that P2P incentive mechanisms are vital for overall system performance. More precisely, as highlighted by Gnutella, solely relying on the altruism of the nodes to contribute their resources will inevitably fail in P2P content distribution systems. Therefore, currently popular P2P systems have explicitly built contribution incentives into their design. Apart from this, it was found that BitTorrent is the most prevalent P2P content distribution protocol currently used, accounting for more than 70% of the total P2P traffic in Germany (which is more than twice the total HTTP traffic). As such, it represents the current de-facto standard for scalable P2P content distribution.

Finally, Section 2.4 classified P2P incentive mechanisms into three major categories: direct reciprocity (bilateral schemes), indirect reciprocity (multilateral schemes), and virtual currency systems. Subsequently, for each category, an overview of the most recent approaches was given. It turned out that the predominant technique to foster cooperation in P2P content distribution systems is the *tit-for-tat* incentive policy, i.e., node A will only upload to node B if node B will upload to node A. This bilateral policy has the big advantage that downloader selection is solely based on direct observations and local histories. This makes it easy to implement and highly robust

to strategic and malicious attacks, as it refrains from using third-party experiences for decision making. However, it must be also noted that a few recent works have challenged the effectiveness of tit-for-tat at promoting long-term cooperation among users. Specifically, they attributed the prevalence of P2P performance and availability problems to it. Analysing and quantifying this important issue will be subject of the following chapters.

3

MEASUREMENTS AND ANALYSIS OF THE BITTORRENT SYSTEM

The previous chapters have criticised P2P content distribution in terms of performance and availability. It has been argued that ineffective incentive design is the core reason for these shortcomings. Before continuing, it is important to validate both these aspects, alongside discovering vital properties that cause these limitations. In this chapter, we perform such an analysis using the BitTorrent system as case study.

3.1 INTRODUCTION

So far it has been discussed that P2P system performance solely depends on the willingness of users to cooperate with the system. Nevertheless, in practise, it can be observed that users often tend to deny cooperation. To tackle this dilemma, current popular P2P content distribution systems predominantly employ tit-for-tat incentive strategies, only rewarding users who have directly contributed previously. Recent work has argued, however, that these strategies can be highly ineffective at promoting long-term cooperation between users, thereby causing serious performance and availability issues.

Therefore, before devising a more sophisticated incentive mechanism, it is first important to *validate* that existing P2P content distribution systems indeed suffer from shortcomings, and that these shortcomings are *caused* by ineffective incentive design. Further, in order to design improved incentives, it is also important to discover *how* existing incentive mechanisms fail to fulfil performance and availability requirements. The research conducted in this chapter primary focuses on these aspects by using the BitTorrent system as a case study. This choice is motivated by the fact that BitTorrent is the most prevalent P2P distribution system currently used. Also, it represents a basic archetype for systems relying on bilateral incentive strategies (i.e., tit-for-tat), thus perfectly satisfying our needs.

We believe the only accurate and valid way to perform such a comprehensive analysis is to run exhaustive measurements in currently deployed BitTorrent swarms. This allows us to infer important low-level details (e.g., overlay swarm characteristics and user behaviour statistics) that cannot be obtained when applying analytical or simulation-based studies using artificial workloads. For that reason, we have conducted two large-scale measurement studies of BitTorrent; the first investigates BitTorrent on a macroscopic level by periodically probing over 46,000 torrents to ascertain their high-level characteristics, such as swarm size and the proportion of leechers and seeders. While, the second study, on the other hand, investigates BitTorrent on a microscopic level by contacting over 700,000 individual peers in 832 torrents to discover relevant properties such as their download rates and chunk availability. To the best of the author's knowledge, this is the largest dataset in terms of size and collected information used to investigate file availability and download performance in BitTorrent.

The rest of this chapter is structured as follows. Section 3.2 details our measurement methodology. After this, in Section 3.3, we utilise our measurement data to study performance and availability issues from a user perspective. Subsequently, Section 3.4 investigates these issues on a system-level to reveal the causes for the shortcomings observed. Finally, Section 3.5 inspects the large-scale effects of these issues, while we conclude in Section 3.6.

3.2 MEASUREMENT METHODOLOGY

In this section, we first give an overview of different techniques to measure BitTorrent-like systems. To this end, we justify the choice of our measurement methodology. Subsequently, we present relevant details about the devised measurement platforms including the scope of our measurements.

3.2.1 *Overview*

The efficiency of BitTorrent is largely dictated by the behaviour and characteristics of the peers operating in the system [100]. Depending on the workload offered, conclusive statements can therefore vary significantly. For that reason, to truly understand availability and performance issues in BitTorrent, it is necessary to *collect* and *analyse* trace data from a large number real swarms, along with the data on why these problems arise in practise. Without this information, any analytical or simulation-based study using artificial workload data as input can only give a rough estimation on what is truly happening in reality.

Unfortunately, as described in Section 2.3.3, the BitTorrent ecosystem consists of dozens of BitTorrent communities, each containing several thousands of torrents. Thus, when performing such a large-scale measurement study, one has to restrict oneself to a single community, since the monitoring of all of them at the same time is near impossible. The measurements conducted and utilised within this thesis therefore solely focus on the *Mininova*[1] community. This choice is motivated by the fact that Mininova was the largest BitTorrent community at the time of the measurements according to the Alexa ranking. Further, Zhang et al. [133] have recently shown that the redundancy of torrent indexing sites is high, and focussing on the biggest BitTorrent community thus provides a representative picture on BitTorrent's efficiency.

The most intuitive way to *collect* this data is to directly access tracker logs. In particular, the BitTorrent protocol specifies that a peer that joins/leaves an overlay swarm is supposed to contact the tracker [21]. Furthermore, it requires that each peer periodically reports statistics such as the amount of data sent and received, and its downloading state. A tracker log therefore contains detailed information about the number of available leechers and seeders, the peers' cooperativeness in terms of uploaded data, and user behaviour characteristics such as seeding times and peer arrival patterns. Unfortunately, however, these traces are often problematic to obtain because they require the agreement from content providers. At best, data of only a few torrents is available [51, 43, 42, 8].

To remedy this, one can inject passive monitoring nodes into each overlay swarm under consideration. These nodes behave similarly to normal BitTorrent users, but do not take part in the downloading process. Instead, they exploit the BitTorrent protocol to continuously request relevant information from the tracker as well as the participating peers. To this end, crawling techniques can be divided into two categories. In its simplest form, a crawler[2] periodically requests IP-addresses from the tracker to learn about other peers participating in the torrent [133, 84]. This makes it possible to study the demographics and population dynamics of the torrents under analysis, what we name *macroscopic crawling*. More sophisticated crawlers also contact the clients and retrieve detailed information such as the client ID and their chunk bitmap (i.e., downloading state). We name this *microscopic crawling*. Although the microscopic crawling gives more detailed information, it is noticeably less scalable and only allows a few hundreds torrents to be studied in parallel [97, 110].

1 http://www.mininova.org

2 In our terminology, a crawler describes a component that can be composed of several physical nodes, each collecting data according to a specific task.

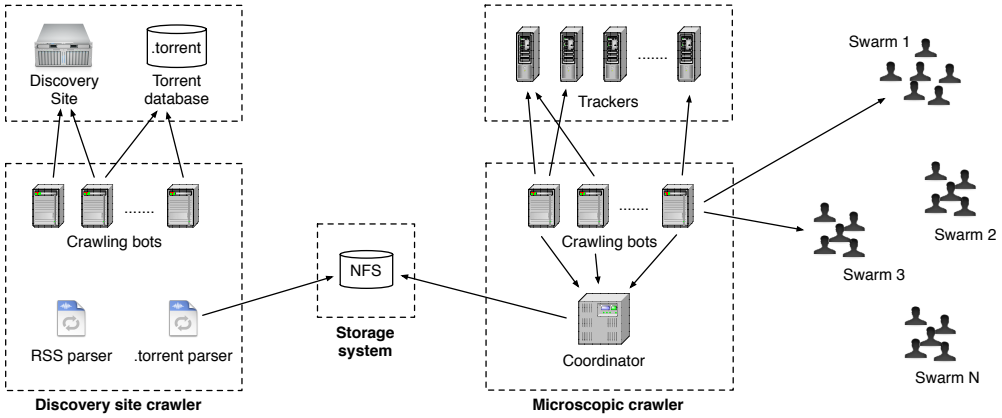


Figure 9: Overview of microscopic measurement platform.

Each crawling technique is effective for addressing particular needs. However, to *analyse* performance and availability issues of BitTorrent in a holistic way, it is important to combine both techniques. For that reason, we have designed and implemented two types of measurement platforms capable of performing microscopic and macroscopic measurements, respectively. In the following, we describe each of both platforms in more detail, along with our measurement methodology.

3.2.2 *Microscopic Crawling*

In this section, we first present relevant details of our measurement platform that can investigate BitTorrent swarms on a microscopic level, using 20 nodes in the Emulab[3] testbed. Subsequently, we give an overview of the scope of our measurements in the Mininova community.

Measurement Infrastructure

The architecture of the microscopic measurement platform consists of two crawlers and a network file system (NFS), as shown in Figure 9. The *discovery site crawler* periodically downloads the RSS web feed document of the Mininova website. By doing so, we are able to learn about all new torrents that have recently been published on Mininova.org. To this end, we developed an XML parser, capable of extracting torrent meta information from the downloaded RSS feed such as the torrent upload time, torrent category, and the download URL of the `.torrent` file. By using this URL, the discovery site crawler then downloads the `.torrent` file, and passes this information to a `.torrent` parser. This parser extracts information such as the torrent creation time, info hash, the IP-address of the tracker(s), data file size, and the number of chunks the file is composed of. All this gathered information is directly stored in the NFS file system for both post-processing purposes and to inform the microscopic crawler about the existence of a new BitTorrent swarm.

The *microscopic crawler* is then responsible for acquiring low-level characteristics from the peers participating in the different BitTorrent swarms. This information includes the peers' chunk bitmaps, their downloading speeds, and the seeding behaviour. To accomplish this, however, a number of challenges must be addressed. First, the crawler must continuously learn about all peers operating in a given BitTorrent swarm. Unfortunately, there is no direct way to retrieve a complete list of these nodes. For instance, querying the tracker that serves a torrent only returns a random pool of IP addresses (usually 50) that participate in the content download. Moreover, the tracker

3 https://www.emulab.net

dictates a fixed time interval (usually around 30-120 minutes) that must pass by until a new request is allowed[4]. Since the population size of real BitTorrent swarms can easily exceed 1,000 online nodes and also varies continuously, using only one physical machine therefore has little prospect of success. Instead, it is mandatory to utilise multiple machines (with different IP addresses) to query the tracker at a much higher time resolution, thereby avoiding the repercussion of being banned.

For that reason, our microscopic crawler consists of one crawling manager and multiple crawling bots. The manager maintains a list of (infohash, tracker)-pairs for all swarms under consideration, as obtained by the discovery site crawler. The crawling bots, each possessing its own IP-address, access this list and simultaneously request information from multiple trackers (cf. Figure 9). This distributed approach allows the system to quickly obtain the IP-addresses for each (infohash, tracker)-pair in a granularity of only a few minutes. To further accelerate the peer discovery process, the crawling bots additionally run the PEX protocol (cf. Section 2.3.3) to learn from already discovered online nodes about the IP-addresses of their neighbours.

After obtaining the IP-addresses of the peers in a given swarm, it is necessary to monitor them, which is even more complicated. In more detail, the only way to infer detailed information about a peer is to establish and maintain a TCP connection to this node. For instance, during the handshaking process, each peer transmits low-level information such as its client ID and chunk bitmap. As soon as the peer successfully downloads a new chunk, it informs its neighbours about this occurrence using HAVE messages [21]. Collecting this data can thus provide a wealth of information, including download speeds and chunk availability in the swarm.

Unfortunately, continuously maintaining TCP connections to users is hardly possible in large-scale crawlings, mainly because of two important issues. First, our goal is to collect data from hundreds of torrents each containing hundreds of peers. Thus, we quickly exceed the maximum number of possible connections per machine (which is typically 1,000). Second, even if we can handle this large number of connections, current BitTorrent implementations quickly drop connections to peers that do not upload data. To remedy this, for each single node, the crawling manager polls every 10 minutes one of the crawling bots in a round-robin fashion. The chosen crawling bot then establishes a TCP connection to this node, requests its chunk bitmap, and immediately disconnects. Through this approach, we are able to monitor several 100,000 peers at the same time, while avoiding being banned as the IP-address of the requesting node permanently changes.

Finally, to identify and determine the uptime of the users, we use <*ip*,*port*>-pairs and assume that a user comes online the first time it is reported/discovered by the tracker/PEX protocol, and stays online until three consecutive connection attempts fail.

Measurement Scope

Our microscopic crawler operated from July 18, 2009 to July 29, 2009 (`micros-1`) and then again from August 19, 2009 to September 5, 2009 (`micros-2`). For the `micros-1` study, the crawler followed 255 torrents appearing on Mininova after the first measurement hour; in these torrents, we observed 246,750 users. The `micros-2` dataset contains information from 577 torrents and 531,089 users. As mentioned above, each dataset consists of the peers' downloading states, seeding and leeching times, and routing table entries sampled with a resolution of every 10 minutes.

Figure 10 shows the geographical origin of the peers encountered in both measurements. Each point in this figure represents a single node. It is noticeable that most of the peers are located in North America and Europe, which is intuitive since most of the content provided on Mininova is in Spanish or English language. Users from China rather access content from Chinese torrent-discovery sites [133]. Nevertheless, as

4 Under-running this time directly results in being banned by the tracker.

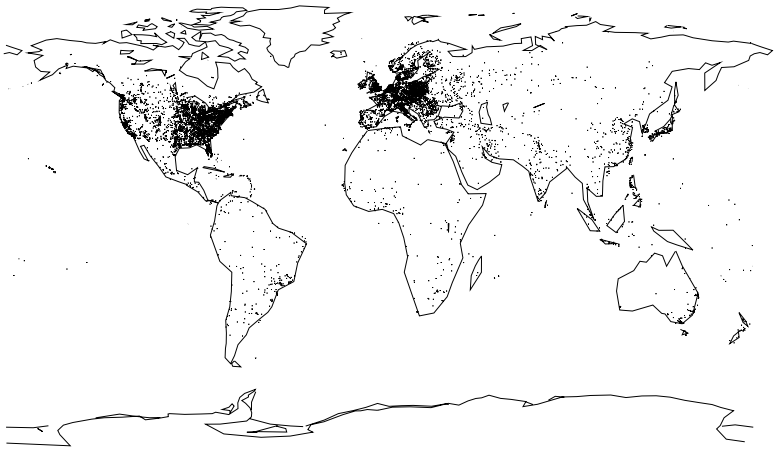


Figure 10: Overview of peer distribution in BitTorrent (microscopic crawling).

demonstrated by the demographics of our macroscopic dataset later on (cf. Figure 12), the geographical distribution of the users monitored in the microscopic crawling is highly representative for the Mininova community, making the dataset very rich for the detailed analysis of user and swarm characteristics in BitTorrent.

3.2.3 *Macroscopic Crawling*

The microscopic measurements provide detailed insights into the distribution of chunks and download rates within a given swarm. However, due to scalability issues, it is difficult to perform such detailed measurements on a very large-scale (e.g., several thousand torrents). To complement these results, we therefore also implemented a higher level measurement platform that followed every torrent published on the Mininova website after December 9, 2008 for a period of 38 days. This study allowed us to gain an extremely large number of measurements regarding details such as peer arrival patterns, seeder/leecher ratios, and torrent sizes. This information can subsequently be correlated with our smaller-scale microscopic measurements to determine the large-scale prevalence of potential performance and availability issues.

Measurement Infrastructure

Our measurement platform consists of four macroscopic crawlers that autonomously operate from different sites in Europe (see Figure 11). Each macroscopic crawler is composed of two sub-crawlers, a discovery site and multi-tracker crawler. The *discovery site crawler* provides similar functionality as the one of the microscopic measurement platform; it periodically downloads the RSS web feed documents of the Mininova website to *(i)* learn about new torrents and to *(ii)* access important torrent meta information. However, the information obtained is stored locally on a hard-disk, instead of using a network file system.

After forwarding the list of newly discovered torrents, the *multi-tracker crawler* repeatedly requests the peer-lists for each (infohash, tracker)-pair. To avoid being banned, the frequency of two consecutive queries is set to the minimum time announced by the tracker (which usually around 15 minutes). However, since all multi-tracker crawlers perform this task at roughly the same time, each tracker under consideration can be requested in intervals of less than 3 minutes on average. This distributed approach enables us to monitor tens of thousands of overlay swarms at the same time, while discovering 98% of the online peers reported by the tracker. Finally, we identify different users by their <*ip,port*>-pairs. Unfortunately, due to scalability issues, we

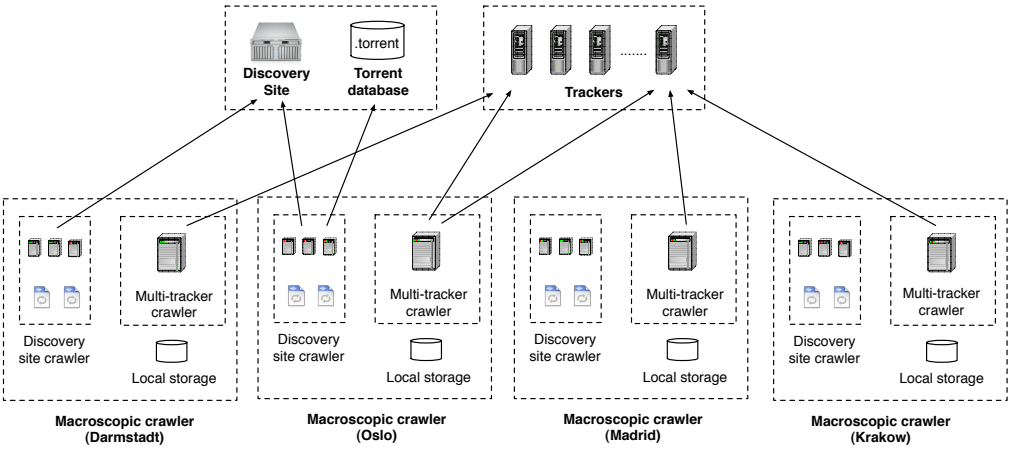


Figure 11: Overview of macroscopic measurement platform.

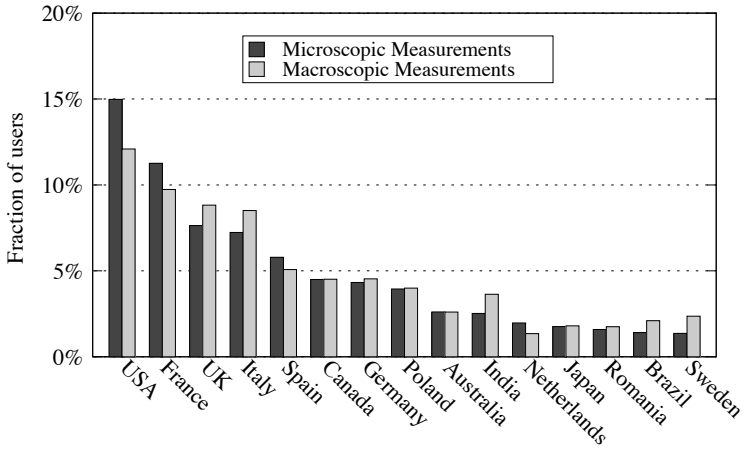


Figure 12: Geographical distribution of all the peers in both types of measurements (top-15 countries).

cannot actively probe these users to determine their session duration. To still have an accurate estimation about their session time, we have devised an analytical model as presented in Appendix A.1.

Measurement Scope

Our macroscopic measurement platform operated from December 09, 2008 to January 16, 2009. Our final dataset consists of reports from 46,227 torrents and 29,066,139 users, giving us large-scale insights into torrent popularities, seeder-to-leecher ratios, user arrival patterns, user session times, and the download interests of the individual user. Figure 12 shows the geographic distribution of the monitored peers. Among the top-15 countries, twelve countries are from North America and Europe, which must be clearly attributed to the language of the content. More importantly, however, it can be noted that the peer distribution of the microscopic and macroscopic crawling is almost identical, indicating that the by an order of magnitude smaller-sized microscopic data set is indeed a representative sample.

3.3 MEASURING USER-LEVEL ISSUES IN THE BITTORRENT SYSTEM

One of the primary research objectives in this thesis is to explore the shortcomings of current P2P content distribution systems, when attempting to achieve high download performance and long-term availability of content. Specifically, we strive to find out what causes these systems to fail to fulfil these properties (cf. Section 1.4). At first, we need, however, to validate that current P2P content distribution systems (i.e., BitTorrent) indeed suffer from these problems. This section focuses on this issue.

It is important to understand that performance and availability can be measured from two different perspectives, namely from the users' perspective and at a system-level. This means, although content can be available in a given BitTorrent swarm (i.e., there are accessible seeders that can serve the file), once performance becomes unacceptably low, users may become frustrated and chose to abort their downloads. Thus, on a system-level, the file is technically 'available', but from the users' point of view, it appears to be 'unavailable'. Intuitively, the most important way to measure both requirements is from the perspective of the users, since their satisfaction determines the success and acceptance of a content distribution system. For the purpose of our analysis, we consider two indicative *user-level* metrics.

The first metric, the *download throughput*, measures performance in BitTorrent. To this end, we are interested in measuring the perceived quality of service over time, rather than focussing on instant download speeds. This is because users typically access the Internet using broadband connections and, thus, expect that their downstream capacity is *continuously* fully utilised. Therefore, we define this performance metric by the total download amount divided by the download time.

The second user-level metric, the download *abortion rate*, quantifies BitTorrent's ability to successfully distribute content to its users, i.e., such that they eventually finish their downloads. Importantly, as discussed above, when users chose to abort downloads, it does not necessarily mean that content is unavailable on a system-level. However, currently, this is sufficient for our needs and, therefore, we delay the exploration of the lower-level details to the next section. The rest of this section now investigates these two user-level metrics.

3.3.1 *Download Throughput*

Over the course of our microscopic crawling, we were able to collect data from 777,839 users, participating in 832 overlay swarms. To determine the performance of these users, we measured the rate over time at which new file chunks appeared in the users' bitfields. Figure 13 summarises the cumulative distributions of download throughputs (the total download amount divided by the download time) for the different BitTorrent users. For comparison purposes, we contrast these throughputs with the cumulative distributions of user downstream link bandwidths as obtained from end-host measurements of 11 major cable and DSL providers in North America and Europe in 2007 [25]. There are two points to notice from this figure:

- First, it can be observed that 8% of the users achieve a download throughput of $\ll 1$ KBps during their entire uptime. This indicates that every 12th user is unable to download content; clearly, this is non-trivial for a content distribution system.
- A second point to notice is that, in general, performance picture is extremely unsatisfying and highly divergent. Specifically, 50% of the users download at a rate of less than 30 KBps. However, most of the users have a downstream link capacity that is significantly higher, ranging from 125 KBps to 1,000 KBps (note that the x-axis is in log-scale). This suggests that overall download performance

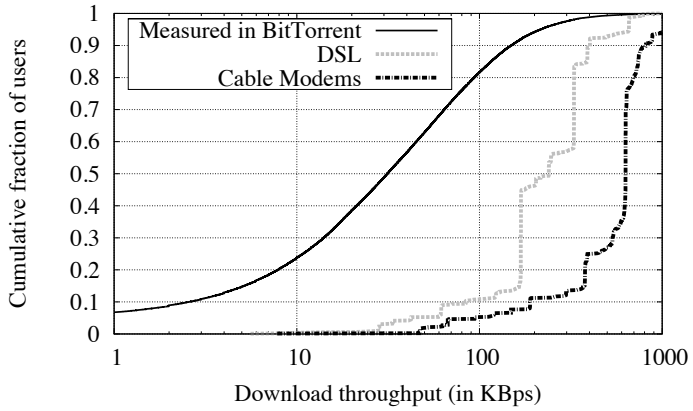


Figure 13: Performance of 777,839 BitTorrent users, participating in 832 overlay swarms. The other two curves show the downstream link bandwidths for the different DSL and cable ISPs, two years before our measurements [25].

as provided by BitTorrent is far from saturating even the modest broadband connections of 2007.

3.3.2 *Abortion Rates*

After inspecting the performance of BitTorrent, we next turn to BitTorrent's ability to distribute a given file to all users in a torrent. Specifically, since we have collected the bitfields' of every node as they have evolved over time, we are able to ascertain whether a user eventually completes its download. Accordingly, we categorise users into two groups: *completed* and *non-completed* users. Table 3 provides statistics for both type of users.

User type	Number of users	Download throughput (in KBps)	Session time (in minutes)	Download progress (in %)	Number of sessions
Completed	423,495	85.91	79.45	100.00	1.23
Non-completed	354,344	33.38	112.05	26.28	1.63
All	777,839	62.54	92.04	65.28	1.40

Table 3: Statistics of users that finished their downloads (completed) and those that did not (non-completed). All information provided are average values calculated over each user group. Session time only considers time in leecher state.

Overall, we see that more than 45% of the users aborted their downloads. Without having further information than the nodes' bitfields, it is difficult to assess the reason behind this large fraction of download abortions. It might be because users *(i)* suddenly lose their interest in the file, *(ii)* download the file in multiple sessions (e.g., shutdown their desktop PC at night), or, most likely, *(iii)* abort their download because of poor performance.

To ascertain this issue, we first consider the session times of users. As can be seen from Table 3, the download sessions of the non-completed users are considerably longer than that of the completed user group. Still, users from the non-completed group retrieve only 26.28% of the file, while the others are able to finish their downloads.

This observation clearly points to the fact that the non-completed users patiently waited for their download completion, rather than losing their interest.

On the other hand, it can also be noted that the reason for high download abortions is not subject to multi-session downloading that could not be captured by our measurement study, i.e., the measurement period was not long enough. This is evident from the fact that the number of download sessions of the non-completed users is even higher than that of the completed users. Instead, there is rather clear evidence that shows slow download rates are the primary reason for this large fraction of download abortions. In fact, as shown in the table, the non-completed users achieve an average download throughput that is 2.5 times lower than that of users completing their downloads.

From these findings, we derive that users are highly sensitive to their perceived instant quality of service. BitTorrent, however, fails to provide sufficient performance levels, thereby inducing an overall abortion rate of more than 45%. Through these findings, we can confidently say that, from the users' perspective, the current BitTorrent system suffers from serious performance and availability issues. Furthermore, the extremely low download rates of some of the users ($\ll$ 1 KBps) give also a clear indication of file availability issues on a system level. Analysing this aspect and identifying the reasons for theses issues will be the subject of the next section.

3.4 CAUSES OF LIMITATIONS: PERFORMANCE AND AVAILABILITY

So far, it has been validated that the current BitTorrent system suffers from shortcomings when distributing content to a set of interested users. To demonstrate this, we have intentionally used higher-level metrics to quantify performance and availability from the users' perspective. This section now focuses on fine-granular system-level aspects, both to characterise and to better understand the causes behind these serious user-level issues.

Previous research (such as [10, 113, 43, 84]) has shown that seeders are vital for high download performance and the long-term availability of content. Further, Guo et al. [43] have promoted the assumption that BitTorrent swarms without seeders are unable to reconstruct the file. In contrast to this, BitTorrent's tit-for-tat incentive mechanism solely applies to users that are *actively* downloading, i.e., it does not provide incentives for seeding (cf. Section 2.4.2). Accordingly, when combining these two aspects, it is intuitive to assume that the user-level issues previously discussed are caused by seeders, particularly by the *lack* of them. Also, this hypothesis is corroborated by the fact every 12th user is unable to download content, indicating the lack of sufficient data sources.

To test and validate this hypothesis, we proceed as follows. By utilising our measurement data, we first confirm the significance of seeders with regard to download performance in BitTorrent (cf. Section 3.4.1). Subsequently, we validate the assumption that a torrent without seeders indeed causes file unavailability (cf. Section 3.4.2). Importantly, in previous research, this is so far an unverified assumption that must be investigated (and quantified). Finally, as a last step in our chain of evidence, we confirm the assumption that due to the lack of appropriate incentives any BitTorrent swarm will inevitably lose seeders over time; a fact that reduces both performance and availability (cf. Section 3.4.3).

To not mix up terminology, when using the term availability in the remainder of this chapter, we always refer to content availability on a *system-level*. This means content is accessible for the users, irrespective of the download rate achieved.

3.4.1 *The Role of Seeders in Download Performance*

In this subsection, we show that seeders are *vital* for achieving high download performance. From this it can then be derived that a lack of them causes serious performance

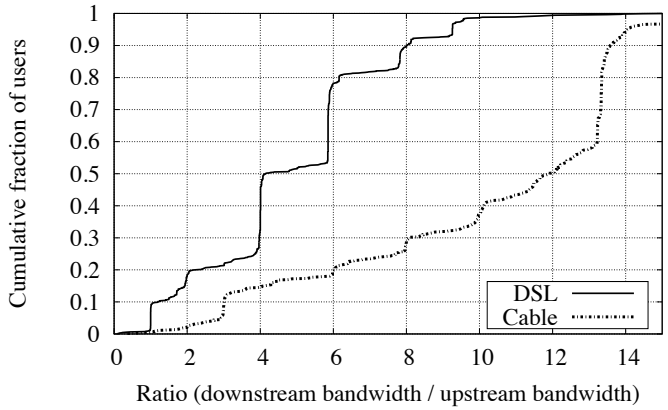


Figure 14: The ratio of downstream to upstream link bandwidths [25].

problems. This is due to the characteristics of users who participate in online content distribution.

The resources of a BitTorrent swarm are solely defined by the peers contributing their upstream capacity. Concretely, the swarm's service capacity U_t is defined by the aggregate of available upload capacity, which can be calculated at time instant t as

$$U_t = \sum_i up_i^S + \sum_j up_j^L. \tag{3.1}$$

Note that up_i^S and up_j^L are the upload contributions of seeder i and leecher j, respectively. On the other hand, the required service capacity D_t at time t is given by the aggregated downstream capacity of all leechers

$$D_t = \sum_j down_j^L. \tag{3.2}$$

Assuming that the users' upload utilisation is always close to optimum (see [10]), and file chunks are only downloaded once, the download links of leechers are saturated if $U_t \geqslant D_t$. Unfortunately, this requirement does not align well with the characteristics of residential broadband networks. More specifically, most of the Internet access links are highly asymmetric with downstream capacities that are often a multiple of the upstream capacity. For instance, Figure 14 plots the ratio of downstream to upstream link bandwidths as obtained from major cable and DSL providers in North America and Europe [25]. For more than half of the DSL hosts, the downstream bandwidths exceed upstream bandwidths by a factor of more than 4. For cable hosts, this factor is even higher with ratios above 10 for more than half of the nodes. Further, it is expected that residential networks will continue to be highly asymmetric, as exemplified by the bandwidths advertised by ISPs on their websites (e.g., [101]).

Due to this high link asymmetry, we must therefore conclude that it is generally impossible for a swarm of solely leechers to achieve downlink saturation, as $\sum_j down_j^L \gg \sum_j up_j^L$. Instead, downlink saturation can only be achieved in the presence of seeders. This happens because seeders increase the available capacity of service without consuming any.

To validate this observation, we make use of our microscopic measurement data. Specifically, we correlate the downlink saturation of peers to the seeder-to-leecher ratio observed in their swarms (the downlink capacity is estimated by gauging each peer's peak download rate). Figure 15 shows the average downlink saturation for all peers based on the seeder-to-leecher ratio of the swarm when they first join (averaged into

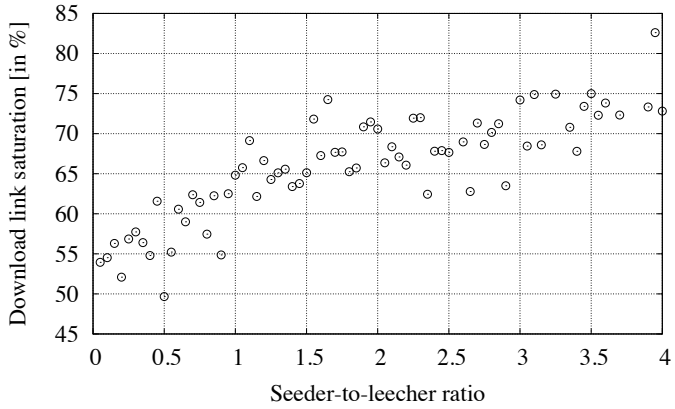


Figure 15: The measured correlation between seeder-to-leecher ratio and downlink saturation.

clusters of size 0.05). In fact, there is a strong trend indicating that the amount of online seeders heavily influences the download performance of the users. For instance, we see that the downlink saturation increases linearly for ratios below 2, indicating a significant performance improvement. As the ratio increases beyond two, however, this trend tails off as many of the lower capacity peers reach their saturation (99% of measured peers had a downlink capacity of under 10 Mbps).

In essence, the above findings have highlighted that seeders are of paramount importance with respect to download performance. Specifically, an overabundance of them are necessary to achieve high download speeds, which is also confirmed by other studies [10, 113].

3.4.2 *The Role of Seeders in File Availability*

So far it has been shown that seeders are vital for achieving high download performance. This was the first step for confirming our hypothesis that the lack of seeders is core reason for the user-level issues previously observed. As a next step, we investigate the role of seeders with regard to file (un)availability. This is important as many users in our microscopic measurement study achieved download rates of less than 1 KBps. Thus, we seek to find out whether these performance issues coincide with the lack of seeders.

Defining Availability

To study and understand the availability of files in BitTorrent, we first present a simple model. Let us assume that we have a torrent T, formed by N nodes, managing the download of a file composed by P chunks. Thus, we can define the vector $V_i = [V_{i1}, V_{i2}, ..., V_{iP}]$ that contains the information about the chunks stored by peer i: $V_{ij} = 1$ if node i has the chunk j; $V_{ij} = 0$ if node i does not have chunk j. V_i is typically known as the *bitfield* of node i.

We define the *percentage of available chunks* of torrent T at a time instant t as

$$U(T) = \frac{\sum_{j=1}^{P} OR(V_{ij})}{P}. \tag{3.3}$$

Where $OR(V_{ij})$ represents the logical *OR*-operation over the chunk j across all the nodes in the torrent T.

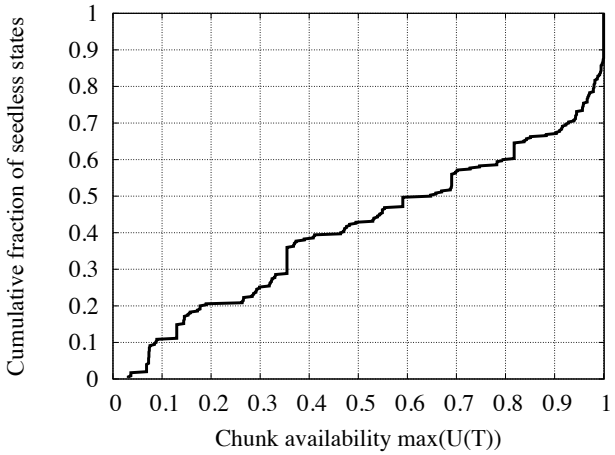


Figure 16: Chunk availability in torrents affected by seedless states.

The Circumstances of Unavailability

It is important to understand in which circumstances a file becomes unavailable, based on our definition. A file is considered unavailable if at least one of its chunks is not accessible within a swarm. This situation arises if there are no peers in the swarm that possess a given chunk, or alternatively, if the peer(s) that possess the chunk are inaccessible (e.g., due to firewalls, NAT, or overlay graph disconnection). It is intuitive to consider the former as a far more likely circumstance (e.g., most BitTorrent clients implement techniques such as NAT traversal [85]. Moreover, they include neighbour discovery techniques such as the Peer Exchange Protocol (PEX) and periodic tracker polling to prevent graph disconnection). Therefore, given this assumption, a file can be considered available if *(i)* there is at least one seeder or *(ii)* there is no seeder but the bitfields of the leechers collectively fit the condition $U(T) = 1$. Without detailed analysis, we can therefore currently state that:

- With an accessible seeder, a file *is* available,
- Without an accessible seeder, a file *may* be available.

We use these two observations as a starting point to investigate unavailability in BitTorrent. In the following, we denote time periods in a torrent's lifecycle in which no seeder is online as a *seedless state*. To this end, the file is *unavailable* if torrent T is in a seedless state and $U(T) < 1$.

It is intuitive to think that $U(T) < 1$ in a torrent without any seeder (that is, leechers alone are unable to reconstruct the file). However, this is, so far, an unverified assumption that must be investigated (and quantified). To ascertain this, the rest of this section inspects the *(i)* nodes' bitfields and *(ii)* nodes' download rates in all the torrents of our microscopic traces affected by seedless states.

Bitfield Analysis

For the purpose of the bitfield analysis, we have collected the nodes' bitfields for all the torrents in our microscopic measurements as they have evolved over time. For each torrent, we have computed $U(T)$ periodically every 10 minutes during any period a torrent is without any seeders (i.e., it is in a seedless state). This allows us to ascertain whether a full copy of the file exists in the torrent at any given time. Figure 16 shows the cumulative distributions of $max(U(T))$ observed in the seedless state for each torrent that we studied.

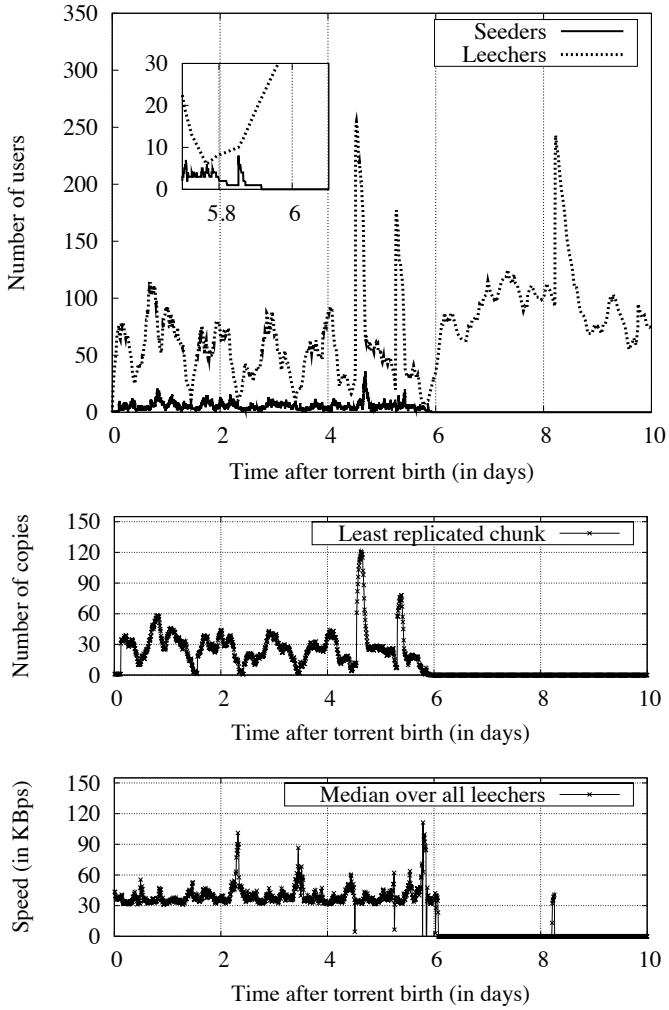


Figure 17: Snapshot from a torrent in our microscopic trace.

As can be seen from the figure, in the majority of cases (86%) our hypothesis is confirmed and the contacted leechers are unable to collectively reconstruct the file once a seeder has left (i.e., $max(U(T)) < 1$). This suggests that the lack of seeders is the primary cause for unavailability of files in BitTorrent. The remaining torrents (14%) are only able to maintain availability due to particular swarm characteristics such as high downloading rates and stable user populations (cf. Appendix A.2).

Download Rate Analysis

A limitation of the bitfield analysis is that not all nodes are accessible to the crawlers due to NATs. To address this, we also inspect the aggregate torrent download rates. Through this we can infer that a file is unavailable when the download rate of all the peers participating in a specific torrent drops close to 0 KBps. From this we can derive that the node cannot find any new chunks to download.

To highlight our findings, we first inspect a representative torrent from our microscopic trace[5], shown in Figure 17. The figure shows the median instant download rate of the online leechers over time, sampled every 10 minutes. It also plots the number of seeders and leechers, as well as the number of copies of the least replicated chunk. Note that when the number of seeders becomes zero, the torrent enters a seedless state.

5 We have observed the same behaviour in most of the torrents affected by seedless states.

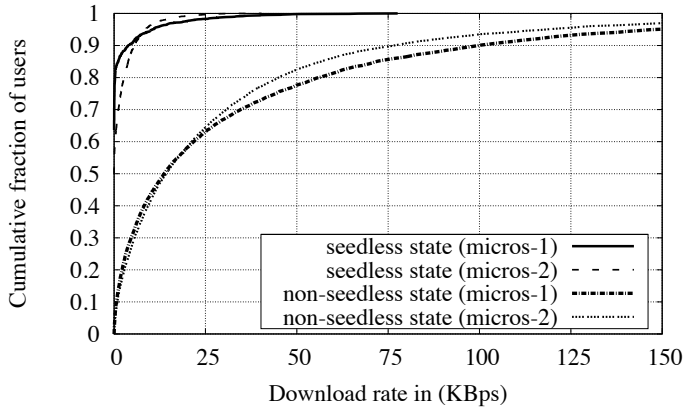


Figure 18: Interval download rates for nodes affected by the lack of seeders.

The torrent can be observed to enter a seedless state before day 6, remaining in this state for the next days. When the final seeder departs, the download rate of the leechers drops to approximately 0-3 KBps after only a few minutes. This also coincides with the number of least replicated chunks dropping to zero. It can therefore be confidently inferred that the file is, indeed, unavailable during this period due to the departure of the last seeder.

The above analysis has inspected a representative torrent. To validate its widespread applicability we also look at the download rate degradation in all torrents. To achieve this, we have taken all the users that have been affected by a seedless state and separated their downloading time into two periods: *(i)* periods in which they have suffered from a seedless state and *(ii)* periods in which they have not. Figure 18 presents the download rate distribution for both periods. First, we can observe that the download rate in a non-seedless state is much higher than in a seedless state. 80-85% of the nodes experience an average download rate lower than 1 KBps when in a seedless torrent, indicating that the peers cannot locate any required chunks and the file is indeed unavailable. Second, however, we also observe that 15-20% of users, in fact, maintain a reasonable level of performance even without any seeders. This can be attributed to two reasons: *(i)* the aforementioned 14% of torrents are capable of reconstructing their file without a seeder at an average rate of 21.3 KBps; and *(ii)* newly joined peers can download the subset of available chunks at an effective rate. This can be observed in the representative torrent (cf. Figure 17): between days 8 and 9 there is a peak in the number of leechers which results in a short peak in the download rate as newcomers download the available chunks.

In sum, we have demonstrated that in spite of the effectiveness of BitTorrent's chunk distribution strategy [70, 71, 10], in the absence of seeders, typically *(i)* leechers are not able to reconstruct a complete file and *(ii)* leechers' download rates rapidly drop to 0 KBps. Therefore, the extremely low download throughput of some users ($\ll$ 1 KBps) must be attributed to the lack of seeders.

3.4.3 *The Emergence of Seedless States*

It has previously been concluded that seedless states generally result in either unavailable content, or, alternatively, low download performance. It is therefore important to gain an understanding of how such seedless states emerge in BitTorrent. We identify two main factors that directly influence the loss of seeders in a BitTorrent swarm: *(i)* the session time of seeders and *(ii)* the inter-arrival rate of the users. To illustrate the influencing factors, we use a simple example shown in Figure 19. In this figure, each

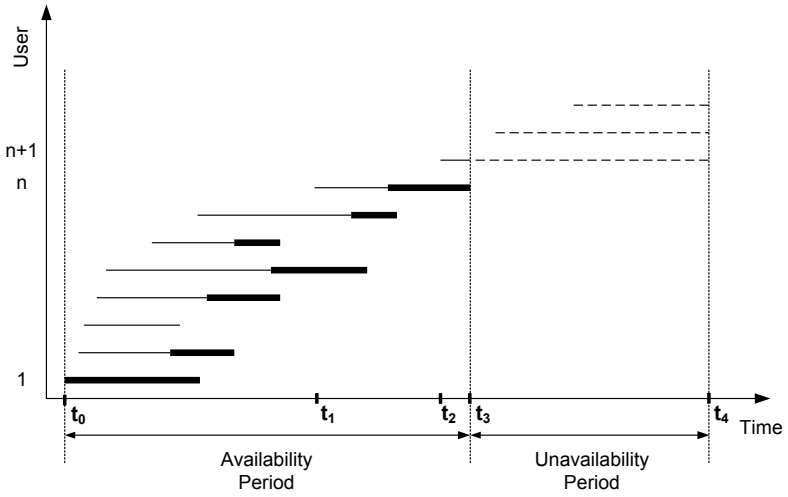


Figure 19: Illustration of a seedless state.

horizontal line represents the lifetime of a user; these users can either be in a leecher state (thin lines) or a seeding state (thick lines).

It seems straightforward that the longer a seeder serves content, the more leechers are able to finish their downloads. As mentioned previously, however, BitTorrent's rate-based incentive strategy only applies in downloading state and only in the context of a single swarm. Therefore, (as demonstrated later on) the seeding times of users are typically quite short, contributing significantly to the frequency of seedless states.

Let us now assume that user n is the last available seeder in our example torrent and none of the previous seeders return to the torrent. In this case, a seedless state occurs when the time required for leechers to download the file exceeds the online time of the last seeder. For example, Figure 19 shows that after the last available seeder leaves the swarm at time t_3, none of the remaining leechers were able to finish the download. If we focus on the n-th node and its subsequent successor in the torrent ($n+1$-th), the inter-arrival time between both users is given by $\tau_{n+1} (= t_2 - t_1)$ whereas the seeding time of node n is given by μ_n. Assume that both users n and n+1 download a file of size F_s with rate D_n and D_{n+1} respectively. Thus, the swarm enters a seedless state when Equation 3.4 is fulfilled.

$$\frac{F_s}{D_n} + \mu_n < \tau_{n+1} + \frac{F_s}{D_{n+1}} \tag{3.4}$$

To simplify the analysis, we assume that $D_n = D_{n+1}$[6]. In this case, the seedless state is reached if the inter-arrival time is larger than the seeding time.

To summarise, seeding times as well as inter-arrival times play an important role in the loss of seeders and subsequently in the long-term availability of content. Since both parameters are not directly correlated, we individually analyse both of them in the following.

Arrival Behaviour of Users

The first behavioural characteristic that is paramount to seedless state generation is the inter-arrival times of users. To this end, intuitive questions are: *(i)* what inter-arrival times do we expect in reality and *(ii)* how do inter-arrival times evolve over time?

By analysing a few hundred torrents in a small community, previous work [42] has shown that user inter-arrival times exponentially increase over time. Our goal is to

6 Our microscopic measurements show that the download rate of users that finish downloads (D_n in the example) is higher than the download rate of those that do not (D_{n+1}) validating our assumption.

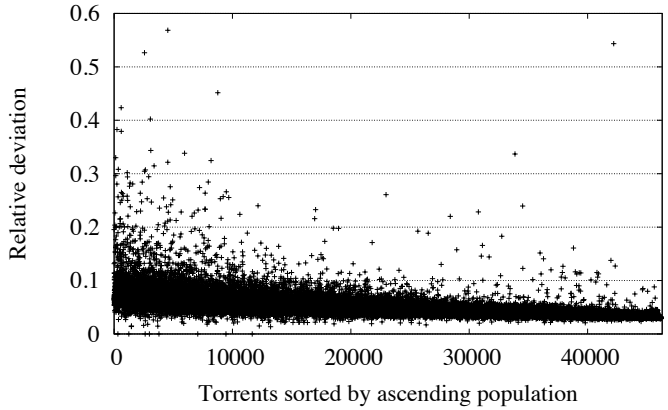


Figure 20: Deviation from linear regression.

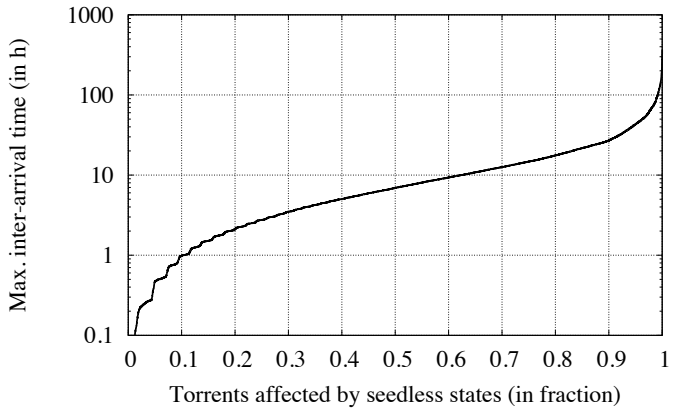


Figure 21: Maximum inter-arrival times of torrents with highly unavailable seeders.

generalize this finding for 'open' communities such as Mininova.org that are orders of magnitude larger. For our analysis, we use similar techniques as applied in [42]. We consider all torrents in our macroscopic trace. We use linear regression to fit the logarithm of the complementary[7] of the number of node arrivals of each torrent along time. Let X_t denote the complementary number of node arrivals at time epoch t and Y_t be the fitting result. We define the relative deviation of the actual node arrivals over an ideally exponentially increasing function by $\frac{\log X_i - \log Y_i}{\log X_i}$. Thus, a relative deviation of 0% indicates that both curves overlap. Figure 20 shows the deviation for each torrent of our macroscopic trace. The x-axis depicts the torrents ordered by ascending population size while the y-axis shows the relative deviation. For most of the torrents, the relative deviation is less than 10% whereas the deviation tends to decrease with increasing torrent popularity. Altogether, the average relative deviation of all torrents is 4.8%. Therefore, we conclude that the inter-arrival time of the nodes exponentially increases with time.

Notably, we observed especially high inter-arrival times in torrents affected by seedless states; this is in line with our analysis of Equation 3.4. For instance, Figure 21 plots the maximum inter-arrival time observed in these torrents with unavailable seeders. More than 50% of the torrents exhibit inter-arrival times far beyond 7 hours.

7 We use the complementary number of node arrivals to avoid domains in which the logarithm is undefined, e.g., epochs with no peer arrivals.

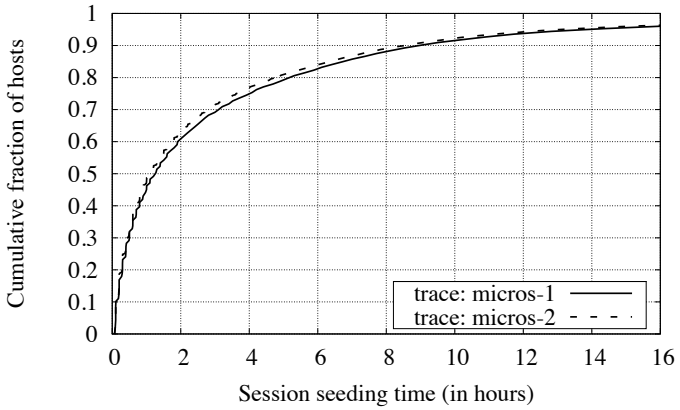


Figure 22: Seeding time distribution.

Seeding Times of Users

The second behavioural characteristic that is paramount to the creation of seedless states is the seeding time of the nodes, i.e., how long seeders stay online for. As already shown in our example torrent (cf. Figure 19), to maintain file availability, it is necessary for seeders to remain online long enough for new seeds to be generated. In addition to this, to achieve high download performance, BitTorrent swarms must be dominated by seeders.

Figure 22 shows the cumulative distribution of the seeding times of the nodes, obtained from the two microscopic measurements. It can be seen that user seeding times are generally short-lasting with 75% of the seeders staying online for less than 4 hours. When this data is compared to the characteristics and length of the inter-arrival time of users, we can deduce two important observations:

- First, since user seeding times are constantly too short and inter-arrival times are rapidly increasing over time, the torrent must inevitably lose seeders. This occurrence can be demonstrated by a representative torrent from our microscopic trace, shown in Figure 23. Upon the initial popularity wave at which many seeders are generated is over, it can be seen that the torrent quickly loses important upload capacity, which clearly reduces performance.
- Second, if we contrast the length of user seeding times (average is 3.44 hours) against the inter-arrival times of users (i.e., 71% of them are $\gg$ 3.44 hours), it can be noticed that these times are not sufficient to avoid seedless states. This fact not only reduces performance, but also prevents torrents from achieving long-term availability of content.

To conclude, through the results obtained in this section, we have confirmed our hypothesis that the lack of seeders is root cause for BitTorrent's performance and availability issues. In particular, the presence of seeders is vital for achieving high download performance and guaranteeing long-term availability of content. However, BitTorrent apparently fails to promote them. This is evident by the observation that current user seeding times in BitTorrent swarms are far too low to ensure the long-term availability of content.

3.5 SCALE OF LIMITATIONS

So far we have validated the existence of shortcomings with regard to performance and file availability using our microscopic measurement data. To extend this analysis and to get a broader understanding of the scale of limitations, we now complement these

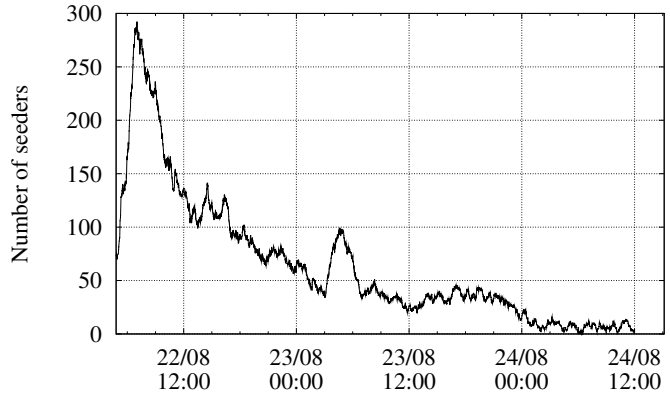


Figure 23: Snapshot of the loss of seeders in a torrent (microscopic measurements).

results with the larger-scale macroscopic measurement data. Through this, it is shown that poor download rates and high abortion levels are common place throughout the entire BitTorrent system.

3.5.1 *The Prevalence of Seedless States*

The previous section has provided a basic understanding on the impact that different seeder-to-leecher ratios have on download performance. In particular, it has been shown that an overabundance of seeders is necessary to maintain high download performance (i.e., to saturate the download links of users). The next step is to contrast this knowledge against the data gathered from our macroscopic crawling. In particular, we are interested in finding out which seeder-to-leecher ratios are common for the current BitTorrent system. This allows us to gain a high-level overview of BitTorrent's performance picture and potential performance constraints.

Figure 24 depicts the cumulative distribution of seeder-to-leecher ratios in all torrents under consideration, one week after the start of our large-scale measurements. From this figure, we can extract two pieces of information; first, more than 17% of the BitTorrent swarms show no active seeders. This suggests that performance in these torrents is solely constrained by *file unavailability*. Also, this coincides with the findings of Section 3.3.1 where several users achieve a download throughput of nearly 0 KBps.

Second, it is clearly noticeable that the amount of seeders in existing overlay swarms is far too low to satisfy the demands of the downloading users. In particular, the majority of swarms (73%) exhibit seeder-to-leecher ratios below 2. However, as shown before, the average downlink saturation in these ranges is only between 49-74%, suggesting poor download rates and significant performance variations.

In general, due to the large variety of seeder-to-leecher ratios ranging from 0.01 to 55, it is now unsurprising that the performance picture of BitTorrent is so divergent. For instance, as shown in Figure 13, 50% of the users achieve download rates far below 30 KBps, whereas a small minority of users (18%) achieve superior performance with rates ranging from 100 KBps to 1 MBps.

To summarise, our macroscopic dataset confirms the existence of large-scale efficiency issues in BitTorrent. Specifically, in some torrents, download performance is constrained by file unavailability. The remaining (available) torrents, on the other hand, are predominantly dominated by leechers, lacking important upload capacity to achieve high download performance.

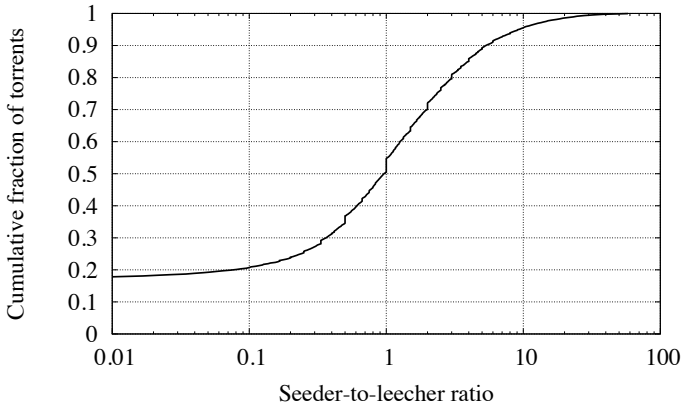


Figure 24: Ratios of seeders to downloading peers for the swarms in our macroscopic measurement study (snapshot after one week). 17.3% of the swarms show no active seeder.

3.5.2 *The Extent of File Unavailability*

The above analysis has revealed that, after only a few days, some BitTorrent swarms show no active seeders. Since the large-scale snapshot of Figure 24 considers several thousands of torrents at a particular stage of their lifecycle (at most one week old), we next focus on the long-term availability of content by considering the entire lifecycle of a torrent (amounting to several weeks). In particular, to quantify how prevalent file unavailability is in BitTorrent, we ask the following question: how *many torrents* and to *what extent* are torrents affected by seedless states? To answer this, we use again the logs from our macroscopic trace that give us a large-scale view on the system comprising of 46,227 torrents.

Our data analysis shows that more than 38% of torrents (17,568 out of 46,227) lose their seeders within the first month, out of which 72% lack seeders after only 5 days. Similarly, we find that more than 45% of the torrents suffer from a lack of seeders for half of their monitoring time. To exemplify the scale of this, in 50% of the torrents observed for periods longer than 30 days, no seeder was available for more than 16 days.

Finally, in our study, more than 9.68 million users participated in torrents with intermittent unavailability. Out of these users, more than 1.59 million were directly affected by the unavailability problem; i.e., they were trying to download content during an unavailable period.

3.6 CONCLUSIONS

In this chapter, we have investigated BitTorrent's efficiency with regard to download performance and file availability. For this purpose, we have conducted two large-scale measurement studies, allowing us to collect data of more than 32 million users, participating in 46,227 BitTorrent swarms. Through the combination of microscopic and macroscopic measurement techniques, we make a number of findings, viz.:

- *Extent of unavailability:* File unavailability is a large-scale problem in BitTorrent. The measurements show that more than 38% of the torrents (17,568 out of 46,227) are affected by unavailability within their first month, out of which 72% are likely to become unavailable after only 5 days. This already affects more than 1.59 million users in the first month of the torrents' lifetime.

- *Poor download rates:* The performance level of BitTorrent swarms is extremely unsatisfying. Specifically, users in BitTorrent download with a rate of only 30 KBps on median, far from saturating even the most modest home broadband connection. Worse, every 12th user is unable to download content at all (i.e., the download throughput is $\ll$ 1 KBps), which is non-trivial for a content distribution system.
- *Poor user satisfaction:* BitTorrent often fails to achieve its overarching goal – to successfully replicate a single file object to all interested users. In particular, due to the low perceived quality of service, user frustration is so high that nearly half of the users (45%) abort downloads.
- *Insufficient incentive design is root cause for the issues observed.* Users that have finished their downloads lack important incentives to persist as file replica in a given overlay swarm. As a result, most of the BitTorrent swarms lose their seeders after only a few days. This occurrence significantly impairs both performance and availability.

Through the findings obtained by these two major measurement studies, the first research goal of this thesis has been addressed (cf. Section 1.4). In particular, by using the popular BitTorrent system as case study, we have demonstrated that current P2P content distribution systems indeed suffer from serious shortcomings with regard to providing continuous content distribution. Also, we have discovered major limitations in existing incentive mechanisms when attempting to achieve these properties — they fail to promote seeding. Seeders are, however, vital for superior download performance and long-term availability of content. Consequently, any viable solution must therefore incentivise these users to persist in the system.

4 ANALYSIS OF SOLUTION SPACE

The previous chapter has outlined the performance and availability issues of current P2P content distribution systems (i.e., BitTorrent) and highlighted the significant impact that seeders have on this. We therefore deduce that a solution must find some way to encourage users to provide content even after they have obtained it themselves. This chapter analysis a set of intuitive solution approaches in an attempt to reduce the solution space and to inform our future design. Through this, a set of key requirements are then outlined to shape the approach taken in the following chapter.

4.1 INTRODUCTION

The effectiveness of BitTorrent has been largely attributed to its rate-based tit-for-tat incentive mechanism that encourages users to contribute resources to achieve higher performance [70, 51, 71]. Despite this, however, we have shown that many torrents do not seem to benefit from this strategy. Instead, a massive proportion of torrents ($\approx$ 40%) achieve extremely low performance with few users being able to download the file successfully.

As shown in the previous chapter, the reason for this significant divergence in performance is what we call the *seeder promotion problem* [60]. This occurs because users are given no incentive to remain online to serve a file after their download has been completed (i.e., to act as seeders). Seeders, however, play a vital role in BitTorrent's performance as they *(i)* provide resources without consuming any, and *(ii)* ensure that a complete copy of the file remains in the swarm. We believe that it is the latter point that is most vital for BitTorrent's performance. Without a fully copy of the file, a swarm is almost always unavailable, even if only one chunk is missing. As such, we consider BitTorrent's performance to be closely linked to its availability.

The solution space for addressing this problem is wide; to assist in its analysis, we categorise solution approaches into two groups: single-torrent solutions and cross-torrent collaboration. The first approach involves users within a *single* swarm cooperating, e.g., to ensure the availability of rare chunks. As such, incentives (such as file encryption or monetary rewards) can be used to encourage users to remain seeding for longer periods of time. The second approach involves the peers of *multiple* swarms cooperating to ensure the availability of rare chunks. As such, cross-torrent solutions aim to build incentives that allow users to make contributions and to receive rewards agnostic to the peers or swarms they interacted with.

Clearly, there is a wide spectrum of potential approaches to address the seeder promotion problem. Therefore, to inform our design, it is important to study these potential solutions from an abstract perspective. The purpose of this is to reduce the solution space to gain an understanding of the best general approach. To achieve this we utilise the trace logs from our measurement studies to analyse the potential of a variety of abstract mechanisms. A further purpose of this chapter is to derive a set of requirements that a solution must fulfil to overcome the performance and availability issues previously discussed. These requirements are intended to be a necessary basis for the solution devised in Chapter 5.

The rest of this chapter is structured as follows; first, in Section 4.2, the key principles of potential single-torrent and cross-torrent mechanisms are briefly outlined to show how each approach might incite users to persist as seeders. Following this, in Section 4.3, these different approaches are quantitatively analysed with regard to three primary metrics: availability, performance and fairness. Based on these findings, in Section 4.4,

we present the deduced solution requirements. Finally, in Section 4.5, the chapter is concluded by summarising our main insights.

4.2 ABSTRACT SOLUTION SPACE

This section briefly outlines the two generic approaches that can be taken for improving seeding in BitTorrent. The first is using traditional single-torrent principles, while the second exploits the concept of cross-torrent collaboration. Note that we do not offer concrete implementational details; instead, we provide a brief outline of the key principles behind each mechanism.

4.2.1 *Single-Torrent Approaches*

A single-torrent solution involves incentivising users to remain within a torrent to act as seeder based on certain properties related to that *individual* torrent. As of yet, we do not know of any successful mechanisms to achieve this. This is due to the difficulty of enforcing incentives once a peer has already obtained the file which it desires. We therefore consider a simple framework of encrypted chunks that may work. Such a solution would involve encrypting the file before it is distributed within the swarm. The tracker would be responsible for managing this encryption, and, as such, would be the source of the keys. Subsequently, once a peer has downloaded the file, it would be required to remain seeding for a length of time determined by the tracker, before the encryption keys are released to it.

As shown in Figure 25, the intuition behind this approach is that, due to the extended seeding time of peer A, a new replacement seeder can be generated (i.e., peer B), preventing a torrent from entering a seedless state. Obviously, to deploy such a solution in practise, several security concerns must be addressed. For instance, it must be prevented that users forward the encryption key to other users in the system by using an alternative channel, e.g., another torrent. On the other hand, this could be easily solved by using a per-peer key management, or, alternatively, the file could be periodically re-encrypted using different encryption keys.

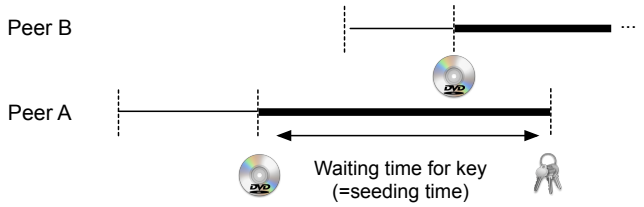


Figure 25: File-encryption to extend the seeding times of the users in a single-torrent approach. Upon peer A has finished its download, it has to wait some time until it can receives the encryption key for the downloaded content. This contributes to the generation of a new replacement seeder, namely peer B.

In contrast to the enforced increase of user seeding times, it is also worthwhile to consider external rewards that are provided by content providers. In such a single-torrent approach, seeders will receive monetary awards for the amount of data they have provided to others nodes in the system. This gives users the freedom to chose how long they want to stay online upon finishing their downloads and is less restrictive than file encryption. In fact, this solution requires that content providers are prepared to spend money for the dissemination of their content files. However, this could be easily accomplished by investing a small fraction of the cost savings due to P2P technology (e.g., bandwidth savings on source servers), resulting in a yet profitable business model.

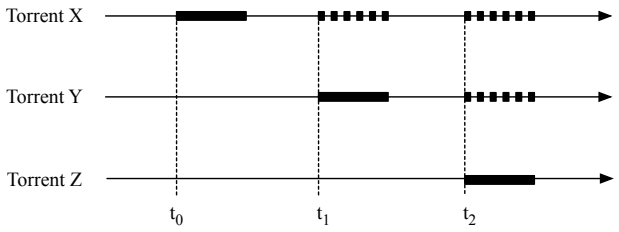


Figure 26: Exemplary lifetime of a peer. The long arrows represent torrents. The bold sections represent intervals when the exemplary peer is active in the corresponding torrents as a leecher. The dashed bold sections represent intervals when the peer could resume seeding in the corresponding torrents in case there were any incentives for it to do so.

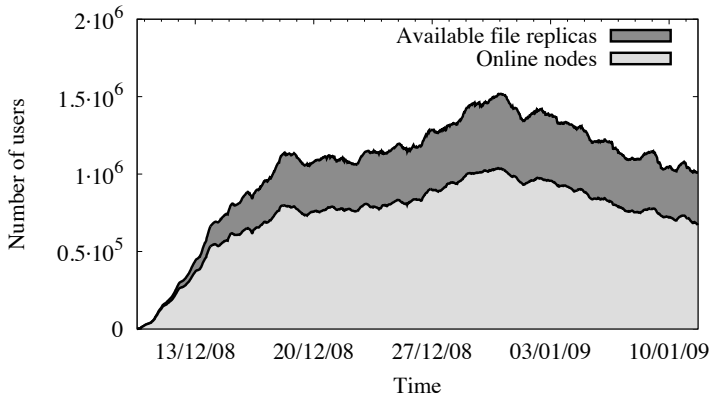


Figure 27: Online users and available file replicas in the category 'Movies' in our macroscopic trace.

4.2.2 *Cross-Torrent Approaches*

An alternative to single-torrent approaches is to find ways of encouraging cooperation between multiple torrents. A cross-torrent solution involves incentivising users to cooperate with the *system* as opposed to individual *torrents*. This approach is motivated by observations from our macroscopic trace showing that 51% of the users join multiple torrents (4.98 on average). We have further found that seeders frequently re-join swarms after they have left, therefore providing conclusive evidence that users re-join the BitTorrent system multiple times while still possessing their previously downloaded files.

To highlight the principles of a cross-torrent solution, imagine a user who joins torrent X at some point in time and completes the download as shown in Figure 26. This user may very well join another torrent Y at a later point in time. When this occurs, the node could also theoretically persist as a *replica* for torrent X as shown by the dashed bold sections in Figure 26. As such, the seeder promotion problem would be addressed by utilising replicas as opposed to traditional seeders (although in practise these are very similar). To inspect the feasibility of this, Figure 27 shows the number of online nodes in the 'Movies' category alongside the number of potential online nodes that could act as replicas. Evidently, there is a large pool of untapped resources that could be exploited; in fact, more than 40% of the online users are indeed file replicas that could provide their previously downloaded content.

So far, it has been shown that there is real-world potential for utilising cross-torrent solutions for addressing the seeder promotion problem. However, as of yet, there exists no deployed solution for incentivising users to cooperate in such a process. In fact, the need to divert upload resources from a node's current torrent would disincentivise cross-torrent collaboration because it would decrease the probability of a node being

unchoked for its own content download. It is therefore important to build robust incentives alongside any cross-torrent protocols. To this end, we next outline three abstract cross-torrent incentive approaches that could encourage users to act as replicas.

Cross-Torrent Bartering

The most straightforward approach is to extend BitTorrent's rate-based tit-for-tat mechanism to operate across multiple torrents. This involves peers bartering with each other for content regardless of what swarm they operate in. This could work as follows: assume that user A has previously downloaded torrent X (fully or partially) and is a leecher in torrent Y. User B, on the other hand, has obtained torrent Y earlier and is now a leecher in torrent X. Both A and B could mutually exchange chunks while still conforming to BitTorrent's tit-for-tat strategy (cf. Figure 28).

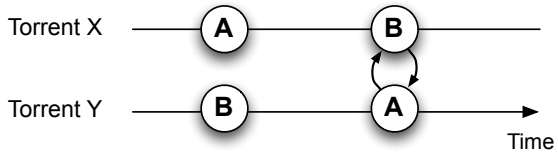


Figure 28: Cross-torrent bartering.

This approach has the advantage that it is instant, based on personal experiences, and does not induce any overhead to exchange information about cooperation across torrents. However, it also has the limitation of needing to locate other peers with shared interests, subsequently restricting the applicability of the approach in any circumstances where such reciprocation cannot be found.

Cross-Torrent Tit-for-Tat

Traditional tit-for-tat and cross-torrent bartering are based on rate-based incentives that are implemented in real-time (i.e., contributions and rewards are instant). An alternative is to base incentives on long-term persistent observations based on total data volume, as exemplified by eMule. In eMule, peers *locally* maintain a persistent history of the contributions made by each user, agnostic to which file and to the time the contribution is made. Subsequently, peers would show a preference to chunk requests from users with higher contribution ratios. As such, peers are encouraged to act as sources for as many files as possible so that they can build up a positive reputation. For instance, as depicted in Figure 29, peer A contributes to B in torrent X because in a future swarm Y, B will recognize A and reciprocate.

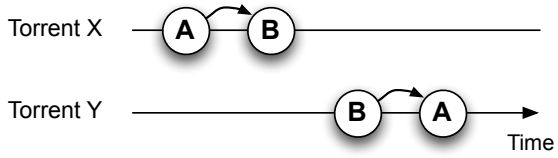


Figure 29: Cross-torrent tit-for-tat (eMule-like).

This approach has the advantage that it might increase the probability of locating shared interests (as with bartering) because incentives become long-term and persistent rather than instant. Importantly, this can also be achieved without introducing any communication overhead or the threat of using third-parties in the process. Unlike bartering, the process is also detached from time, thereby allowing peers to claim back contribution at a later date. However, persistent histories still require repeated interactions between peers, possibly resulting in restricted applicability.

Cross-Torrent Indirect Reciprocation

The previous two approaches rely on direct observations that are stored locally. An alternative is to use persistent history information that is agnostic to individual peers. As such, peer A would be able to make a contribution to peer X and receive the reward from peer Y seamlessly (cf. Figure 30). This solution approach would require some form of reputation infrastructure that can reliably store information about a given peer's 'balance'. For instance, for the approach to work, peer Y must see the contributions of peer A in the entire system. This could be realised by using a shared contribution history in which peers exchange information about their local experiences with other users in the system. As highlighted in Section 2.4, virtual currency systems or indirect reciprocation schemes can fulfil this condition.

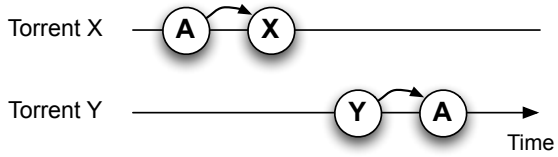


Figure 30: Cross-torrent indirect reciprocation.

Cross-torrent indirect reciprocation has the advantage of detaching incentives from time, torrents, and individual peers, thereby offering the 'purest' form of cross-torrent collaboration. This subsequently addresses the need for synchronous interests (bartering) or repeated interactions (tit-for-tat). However, such an approach also introduces far greater complexity and overhead into the system, potentially negating these benefits in certain environments.

4.2.3 *Summary*

For the sake of clarity, Table 4 gives an overview of the properties of the single-torrent and cross-torrent solution approaches presented before. Most of the properties are straightforward. Two, however, require some explanation: the identifiers and the communication overhead. To enable long-term histories about the users' share ratio, permanent identifiers are required to identify users across different downloading sessions. Contrary to this, temporary identifiers are newly generated whenever a peer joins the system and can, thus, only be applied for incentive schemes where reciprocation for contributions occurs in real-time. From the users' perspective, the latter is always preferable as temporary identifiers provide privacy, e.g., users cannot be tracked in their long-term download behaviour. Communication overhead, on the other hand, indicates whether users must exchange messages. Evidently, indirect reciprocation is the only approach that requires network communication, because peers evaluate each other based on local observations, and, additionally, on third-party experiences.

As mentioned previously, single-torrent approaches strive to extend the seeding times of users, immediately after they have finished their downloads. Consequently, these approaches can never be agnostic to time, torrents, or individual users. Instead, incentives to seeding are either enforced by file-encryption (no reward) or provided with external monetary awards from content providers.

From the category of cross-torrent solutions, cross-torrent bartering is also neither agnostic to time, torrents, or peers. Instead, for the approach to work, synchronous interests between the nodes are required. However, it is most inline with BitTorrent and could, thus, directly be deployed in the current BitTorrent system. Cross-torrent tit-for-tat, on the other hand, detaches from time and torrents, but is still tied to individual users. In particular, it requires that peers repeatedly meet each other in

Solution Category	Single-Torrent	Cross-Torrent		
	File encryption, monetary awards	Bartering	Tit-for-tat	Indirect reciprocation
Incentive	bilateral	bilateral	bilateral	multilateral
Ranking of peers	rate-based	rate-based	data volume	data volume
Information horizon	local	local	local	shared
Reward	no / monetary	instant	long-term	long-term
Identifiers	temporary	temporary	permanent	permanent
Detaches time	no	no	yes	yes
Detaches torrents	no	no	yes	yes
Detaches peers	no	no	no	yes
Comm. overhead	no	no	no	yes

Table 4: Overview of abstract approaches for seeding in BitTorrent.

different torrents such that reciprocation for previous contributions can take place. Still, no communication between the nodes is needed to realise the approach, but permanent identifiers have to be added. Finally, cross-torrent indirect reciprocation is the least restrictive; it even allows peers to claim back contributions at any time from any peer. This comes, however, at the expense of others factors, including higher complexity, management overhead, and network communication.

4.3 QUANTITATIVE ANALYSIS OF ABSTRACT SOLUTION SPACE

In this section, we perform a quantitative analysis of the different abstract solution approaches. Importantly, we do not aim to perform an implementational comparison between the different approaches of the abstract solution space, e.g., regarding protocol overhead, security concerns as well as technical aspects to realise either approach. This is out of the scope of this thesis. Instead, we aim to reduce the solution space by generally exploring the potential of either approach with regard to file availability, download performance, and fairness. We consider fairness as also important because performance must not be achieved at the expense of this property, as discussed in Section 1.3.

To enable this comprehensive analysis in an accurate way, we perform extensive trace-driven simulations using the data acquired by our crawlers. Specifically, to benchmark the potential of the different abstract solution directions, we use as a baseline vanilla BitTorrent as deployed in real BitTorrent swarms today.

In the following, we first present important details of our evaluation methodology to accomplish this analysis. With this information we then proceed to explore the effectiveness of the different approaches based on the three metrics: availability, performance and fairness.

4.3.1 *Experimental Methodology*

Exploring the effectiveness of strategies to promote long-term incentives for seeding – a key requirement for improving file availability and performance in P2P content distribution systems – requires a workload model that is itself long-term. In particular, to truly quantify the impact of novel incentive strategies in the BitTorrent system, it is important to reflect a BitTorrent community in lifelike manner, and, ideally, over a prolonged period of time. To meet this requirement, we exploit the long-term

measurement data acquired by our BitTorrent crawlers and use it as an input for trace-driven simulations.

We next justify the choice of our simulation tool, capable of enabling such trace-driven simulations. As the chosen simulator had to be adapted to our specific needs, we also describe the changes that we have made and how we validated the correctness of the results produced by this framework. Lastly, we detail how we have modelled the different incentive mechanisms in the simulator and how the experiments have been configured using our measurement data.

Simulation Tool

For our trace-driven analysis, we use the Octosim BitTorrent simulator of Bharambe et al. [10], developed by the Carnegie Mellon University and Microsoft Research. Our choice is motivated by the fact that this simulator is also used by other research groups for BitTorrent research, allowing comparability between the experiments. The simulator is discrete event-based and models many of BitTorrent's mechanisms in detail. For instance, it accurately models BitTorrent's chunk selection strategy and unchoking algorithm, including data flows on the transport layer. For further details about the simulation framework, we refer the interested reader to [9].

To adapt the simulator to our needs and to further increase the realism of the simulation model, we have modified the simulator as follows:

- We enable the simulator to support representative bandwidth distributions taken from real-live measurements.
- We extend the peer arrival model to support file request patterns that follow our trace data.
- We integrated functionality that a given user stays online as long as dictated by our trace logs.
- We implement the optimistic disconnect functionality as used in current Azureus implementations[1].
- We update the choking algorithm in seeder state according to recent protocol changes as reported by Legout et al. [70].
- We extend the simulator to enable cross-torrent collaboration and to support the simulation of multiple torrents at the same time.

Simulator Verification

A simulator is useless if the results it produces do not coincide with the reality. Modelling the complexity of BitTorrent's mechanisms as well as transport layer functionalities (e.g., TCP) requires a certain degree of abstraction to allow scalability. To verify how accurate the forecasted simulation results are compared to live-experiments, we have re-run various setups from a well-known BitTorrent measurement study of Legout et al. [71]. In this work, the authors perform a wealth of experiments in the PlanetLab testbed to explore the characteristics of important mechanisms associated to BitTorrent. These experiments include a study of BitTorrent's effectiveness to find peers with similar bandwidth capacities, the interdependency between upload contribution and download reward in BitTorrent, and the importance of the revised unchoking mechanism in seeder state [71].

Clearly, if a simulation model is unable to accurately capture these important characteristics, conclusive statements can be highly misleading. For this reason, by using the same workload settings as presented in [71], we have repeated each of these

1 We find this important as Azureus is the most commonly used client in our measurement study.

experiments with the modified OctoSim BitTorrent simulator. The detailed setup for each experiment can be found in Appendix A.5, including the comparative results. In general, we can confidently say that the outcomes produced by the simulator are sufficiently comparable to those obtained by real-world experiments.

Incentive Model

Since the goal of our analysis is to capture the potential of the different solution directions, we abstract away from technical constraints and assume global knowledge for approaches requiring additional information from the system. This is sufficient for our needs, even though the results produced represent an upper bound for a solution's performance capabilities.

The single-torrent solutions seek to extend the seeding times of users, once their downloads have completed. To study the impact of this in the BitTorrent system, we assume a single-torrent approach capable of extending the measured seeding times of the users by a factor of 2, 5, and 10. This is representative for the proposed file encryption framework or content providers providing monetary awards.

Cross-torrent bartering requires that users locate others in the system that own content (or parts of it) that they desire, and vice versa. In our simulator, it is assumed that these users always find each other (if simultaneously online) and can also always establish TCP connections to each other to mutually exchange data.

Cross-torrent indirect reciprocation, on the other hand, requires that users can see the contributions of their interaction partners to other nodes in the system. Ideally, they can see all contributions of them, across time and torrents. Therefore, when encountering a new user, the contribution history of this node is globally visible and thus directly known.

Finally, in the volume-based cross-torrent approaches (i.e., cross-torrent tit-for-tat and indirect reciprocation), we assume that users always prefer peers with the highest share ratios for uploading.

Workload Model

We now describe the input parameters for the trace-driven simulations. This comprises the torrents we selected from our trace data, the user behaviour model derived from these torrents as well as how we estimated the speed of the Internet connections of these users.

- *Selecting the torrents.* Our trace data encompasses tens of thousands of torrents over a period of several weeks, far more than the simulator is able to handle. Hence, we choose a random subset of a hundred torrents from the set of torrents affected by seedless states. These torrents vary in file sizes between 3-1,500 MB and exhibit a per-torrent monitoring period of at least four weeks. This procedure has been repeated five times with different torrent subsets. The logs of each of these subsets contain data of more than 45,000 downloads.

- *User behaviour.* To model the access pattern of torrents, we do not use any artificial peer arrival function. Instead, we bring up peers according to the trace logs. To model the number of swarms that a peer joins, we calculate the probability distribution over our entire data set. Any user that cannot download the file within 36 hours aborts the download[2]. The session time of a user consists of a busy and idle period. In *busy* period, the user is actively downloading until it reaches seeder state. Subsequently, the *idle* period begins in which the user remains until it quits the client. During this idle time, in the single-torrent

2 We find through simulations that 36 hours is enough time to get a download success ratio over 99% in the presence of seeders for all access links and file sizes used in our experiments.

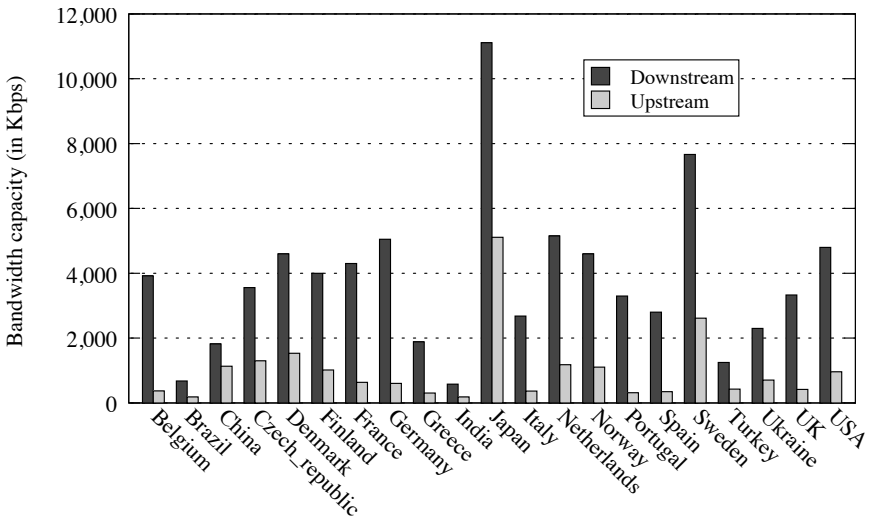


Figure 31: Excerpt of statistics of median country speeds (Ookla database [121]).

approach, the user serves the downloaded content file to other nodes in the overlay swarm. While, for the cross-torrent variants, the user additionally serves downloaded content files in multiple overlay swarms, respectively. Finally, to realistically model these idle periods for users, we use the measured seeding time distribution of BitTorrent users, with an average of 3.37 hours (see Figure 22).

- *Speed distributions.* To realistically reflect the Internet access connection of the different users contained in our trace logs, we first associate each IP address with a country using a freely available geolocation database [82]. Based on the country of origin, the Ookla database provides us with the median down/uplink capacity of each user [121]. Figure 31 gives an overview of statistics of the speed distributions on a per-country level, as obtained from this data set.

4.3.2 *Formative Results*

We now present the formative results of the different possible single-torrent and cross-torrent approaches based on three metrics: *availability*, *performance* and *fairness*. For conciseness and ease of representation, we focus on the results obtained from one out of five torrent subsets. The results and conclusions drawn of the other subsets are, however, similar.

Availability

A file is considered unavailable if at least one of its chunks is not accessible within a swarm. This situation often coincides with a lack of seeders as seen in Section 3.4.2. It means that any users attempting to download the file will fail; a prominent metric for measuring this is the abortion rate as most users are only prepared to wait a limited length of time during an unavailability period.

Figure 32 shows the fraction of users that abort their downloads when utilising the different approaches. In addition, Table 5 gives an overview of the idle times of users and their seeding times on average. Note that in the cross-torrent variants, the idle time obviously differs from the seeding time. This is because cross-torrent collaboration allows users to seed in two or more torrents while being in busy and/or idle state.

The simulations show that 19.67% of downloads were not successful in vanilla BitTorrent. This confirms our observation that nodes do not remain as seeders for long enough to overcome the exponentially increasing inter-arrival times of users. Worse,

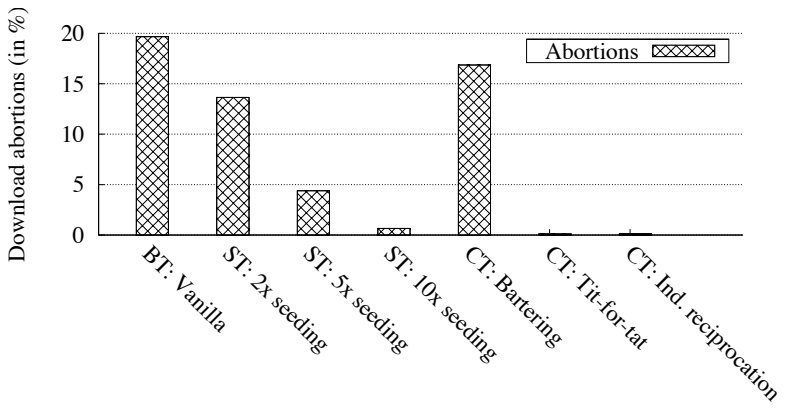


Figure 32: Fraction of download abortions of the different approaches.

Variant	User stats		Metric	
	Idle time (in hours)	Seeding time (in hours)	Down. throughput (in KBps)	Abortion rate (in %)
BT: Vanilla	3.37	3.37	145.57	19.67
ST: 2x seeding	6.88	6.88	186.28	13.65
ST: 5x seeding	17.20	17.20	249.95	4.39
ST: 10x seeding	34.40	34.40	274.59	0.66
CT: Bartering	3.37	4.84	127.93	16.86
CT: Tit-for-tat	3.37	10.36	76.45	0.11
CT: Ind. reciprocation	3.37	9.51	196.81	0.13

Table 5: Overview of results of quantitative solution space analysis. All presented figures are average values, except the abortion rate.

due to extremely long inter-arrival times (often exceeding 10 hours), a single-torrent approach even capable of increasing seeding times by a factor of 2 or 5 is limited in its success. To maintain persistent file availability (i.e., success rate >99%), the users must therefore stay on average 10 times longer after downloading. As such, to achieve availability, vanilla BitTorrent would require average seeding times of more than 34 hours.

The first cross-torrent approach inspected is *cross-torrent bartering*. The results show that this also fails to significantly improve availability. In fact, there is only a 2.78% improvement over vanilla BitTorrent. This occurs because cross-torrent bartering assumes that large numbers of peers operate in swarms with synchronous interests. The trace-based simulations show that this is, in fact, not an accurate assumption. The measurement study results also corroborate this finding; these show that the probability of bartering working in the real-world is below 0.1%. Therefore, the circumstances in which users can act as file replicas are very seldom.

In contrast to these results, the other two cross-torrent approaches (tit-for-tat and indirect reciprocity) are able to effectively maintain persistent file availability. As opposed to cross-torrent bartering, these solutions do not require immediate reciprocation. Instead, peers can claim back their rewards in the future and are thereby encouraged to act as a file replica in the hope of later gaining an advantage. In the case of cross-torrent tit-for-tat this involves repeat interactions, while in the case of indirect reciprocation this involves interacting with any peer. This approach of detaching incentives from time therefore perfectly addresses the availability issue.

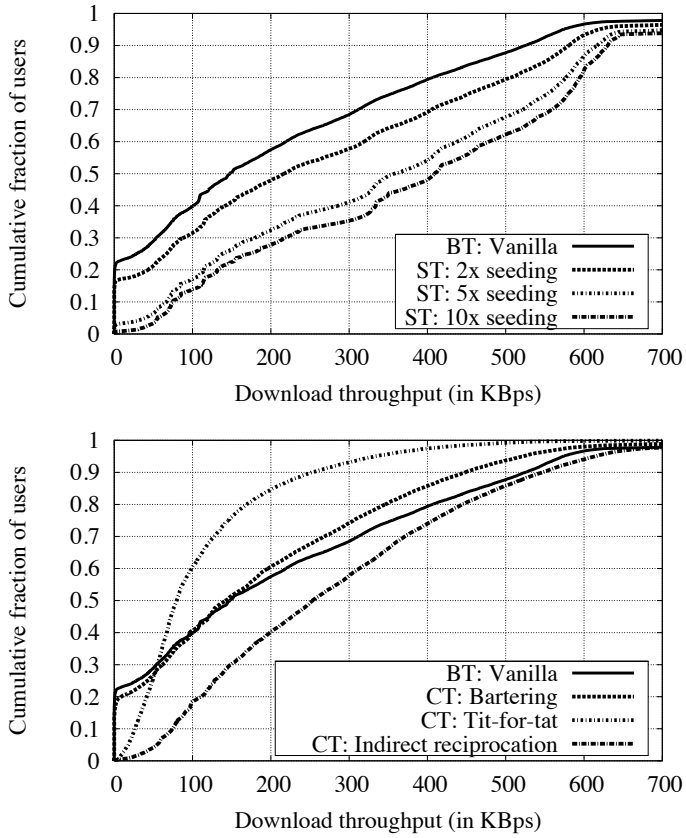


Figure 33: Cumulative distribution functions of the measured download rates. Top: Variants with lengthened seeding times (Single-torrent approaches). Bottom: Cross-torrent approaches.

Performance

While some solutions have been shown to enable persistent availability, we have also found that users are highly sensitive to their perceived instant quality of service (cf. Section 3.3.2). Therefore, any solution must also maintain an acceptable download rate while improving availability. To study this, the average download rates in each torrent have been recorded when utilising the single-torrent and various cross-torrent solutions. Figure 33 shows the cumulative distribution of these download rates.

It can first be observed that roughly 20% of the downloads in vanilla BitTorrent are below 1 KBps. As such, it can be considered that performance is unacceptably low. The reason for this is the poor availability observed in the 20% of torrents as discussed in the availability analysis. Clearly, both availability and performance in BitTorrent are inexorably linked: torrents that are unavailable also have low performance. Of course, as previously shown, this problem can be addressed in a single-torrent approach by extending seeding times, thereby ensuring availability. The increase of seeding times by a factor of 2, 5, and 10 also has the added benefit of increasing swarm resources. As exemplified in Table 5, the single-torrent variant with 10x enlarged seeding (34.4 hours on average) achieves by far the highest average download throughput of all abstract solution approaches, confirming the findings of previous chapter that seeders are also vital for download performance.

Considering the results of our availability analysis, it is unsurprising that cross-torrent bartering also does not offer significant performance benefits. This is because it is essentially the same as vanilla BitTorrent but with the ability to operate across different torrents. It has previously been shown that this does not really improve

availability, and, consequently, results in poor performance due to the dependency of performance on availability.

The availability analysis has found that cross-torrent tit-for-tat does improve availability and, therefore, it is logical to assume that performance is also improved. In fact, this approach does result in a significant increase in nodes that find content available (>19%). However, the simulations show that this does not translate into performance improvements. Instead, users can access the files with high availability but with poor performance. This is consistent with most people's daily experience of using the eMule application (which actually employs volume-based cross-torrent tit-for-tat). The reason for this is that eMule relies on repeated interactions by maintaining persistent records. A peer that contributes resources to another peer can therefore only recoup them if there is a later repeated interaction. This therefore creates incentives for sharing, but prevents a peer from claiming back contributions from an arbitrary peer on many occasions.

The last cross-torrent solution inspected is cross-torrent indirect reciprocation which has already been shown to vastly improve availability. The results show that, unlike cross-torrent tit-for-tat, this actually does translate into superior performance. In fact, data inspection reveals that 54% of the users would gain a performance boost of a factor of more than 4 when switching from the eMule-based approach to indirect reciprocation. The reason is that indirect reciprocation allows users to make contributions and to claim them back from any user and any torrent without the need for repeated interactions. This means that a peer will receive superior performance from any peer if it, in return, offers resources to the system as a whole.

Fairness

Whereas the previous two evaluative metrics have looked at aspects that are vital for the continued success of BitTorrent, a further property that would also be desirable is *fairness*. This is defined by the amount of reciprocated data generated by contributions. For incentives such as applied in vanilla BitTorrent or cross-torrent bartering, reciprocation is immediate and can therefore be directly measured. For persistent contribution histories, however, peers may experience lengthy delays before receiving reciprocation. Simply considering a snapshot of the users' share ratio can therefore be a misleading measure, because in the future, this measure may change. For this reason, we consider the following fairness criteria, as defined by Legout et al. [70]: any peer x with an upload rate U_x should get a higher download rate than any other peer y with an upload rate $U_y < U_x$.

To quantify the relation between the upload rate of users (U) and their experienced download rate (D), we compute the correlation coefficient $\rho_{U,D}$ over all users that join *(i)* just a single torrent and *(ii)* multiple torrents, as shown in Figure 34. In particular, differentiating users in these two groups allows us to quantify whether users persisting as file replicas benefit from their behaviour.

The single-torrent variants (vanilla and extended seeding times) show a positive correlation between U and D. In fact, their correlations are very similar suggesting that they all offer a similar level of fairness. Clearly, reciprocation in BitTorrent is based on immediate rate-based observations, and, as such, it is not surprising that the peer selection strategy does an effective job of matching users with similar capabilities [70]. Similarly, cross-torrent bartering offers a fairness level that is largely identical to vanilla BitTorrent. This is intuitive as it operates using BitTorrent's peer selection strategy with the added capability of being able to interact with peers in different torrents.

In contrast to these results, cross-torrent tit-for-tat (the eMule-like approach) exhibits at best a weak correlation for single torrent users and no correlation at all for multi-torrent users. This suggests that users see poor returns when acting as file replicas. The reason for this is twofold. First, the need for repeated interactions means that sometimes a peer will make a contribution without ever receiving any benefit in the

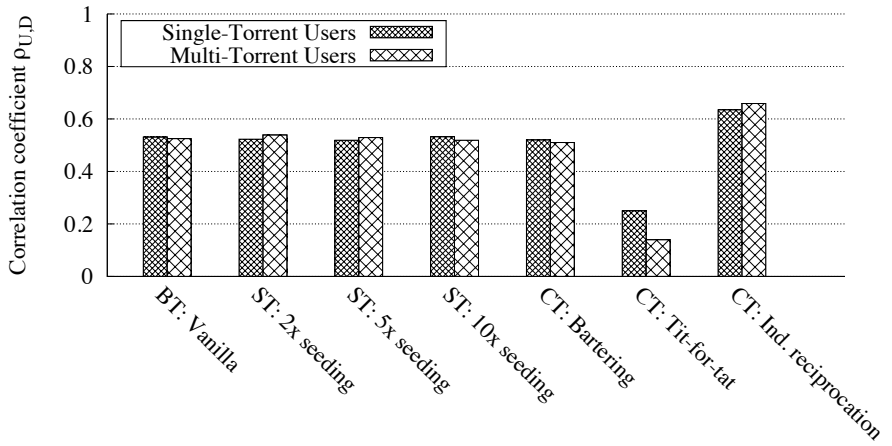


Figure 34: Correlation between upload contribution (U) and download reward (D) for single-torrent and multi-torrent users.

future. This can occur due to permanent peer departures, or, alternatively, due to bad luck on the part of the contributor. In fact, within the measurement study, only 19% of users ever meet each other repeatedly resulting in 81% of contributions being unclaimed.

Last, cross-torrent indirect reciprocation offers the highest level of fairness, even outperforming vanilla BitTorrent. However, when using indirect reciprocation, a peer makes unchoking decisions based on the globally recorded share ratios of any requester (as opposed to rate-based). This shows that using share ratios is at least equally effective at achieving fairness as the traditional approach of using observed upload rates.

4.3.3 *Summary*

In general, our trace-based simulation study has confirmed what we have already discovered in our large-scale measurements: availability and performance are very closely linked with unavailable torrents also being low performance torrents. As such, the simulations show that both vanilla BitTorrent and cross-torrent bartering fail to offer high performance because they fail to improve availability.

Our experiments also show the fundamental limitations of single-torrent-based seeding time extension, e.g., with the aid of file encryption or monetary awards. Here, the seeding times must be extended by an extreme length (more than 34 hours) to achieve high availability and performance. The reason for this is the exponentially increasing inter-arrival times that can quickly exceed 10 hours after only a few days (cf. Section 3.4.3). As such, we consider any solution that seeks to extend the seeding times of users in a single-torrent approach as highly impracticable, as it is likely that users will refuse to remain online for several days, every time they download a file of only a few megabytes.

Similarly, cross-torrent bartering does not offer any real solution for the availability problem due to the low probability of successfully finding peers with synchronous interests. This results in low download performance and high abortion rates.

In contrast, both cross-torrent tit-for-tat (the eMule-like approach) and cross-torrent indirect reciprocation incentives offer extremely effective mechanisms for addressing unavailability, even outperforming the costly extension of seeding times. This is because peer contribution becomes detached from time, allowing peers to claim back contributions at a later date.

Interestingly, cross-torrent tit-for-tat improves availability without also improving performance. This is because a peer can only recoup its contributions through repeated interactions. It is impossible, as such, for a peer to gain superior performance unless it

re-encounters a past peer. In contrast, indirect reciprocation offers the best performance by a significant margin due to its ability to incentivise peers to make contributions to any and all torrents, confident in the knowledge that this strategy can improve their own position.

Finally, the single-torrent approaches as well as the different cross-torrent variants all have positive fairness characteristics with the exclusion of cross-torrent tit-for-tat. This is because cross-torrent tit-for-tat makes unassured investments that may not be recouped in the future. This is an endemic problem of any local persistent history mechanism. Therefore, in practise, long-term users of cross-torrent tit-for-tat (e.g., eMule/eDonkey and its variants) are likely to reduce their sharing, thereby undermining its previously identified benefits.

In summary, we conclude that cross-torrent indirect reciprocation outperforms other approaches regarding the combination of all three metrics: availability, performance, and fairness.

4.4 CONCRETE SOLUTION REQUIREMENTS

Based on the findings of the solution space analysis, we next deduce a set of key requirements that a concrete solution must fulfil to address the seeder promotion problem, as highlighted in BitTorrent. Importantly, these requirements take also implicitly the research challenges presented in Section 1.3 into account.

4.4.1 *Multilateral Incentive Design*

The results of the previous section suggest that the use of cross-torrent bilateral incentives is just as ineffective as single-torrent bilateral incentives at solving the seeder promotion problem. This has been found to be true even when incentives are detached from time. Instead, a solution requires a *multilateral* incentive strategy that is not only detached from content and time, but also from individual users. As such, a user can contribute data to a peer X at a given time and receive the reward from another peer Y at another point in time without encountering this node before.

Interestingly, such an incentive strategy also enables that even lower provisioned nodes can achieve high download performance; a fact that is only hardly possible in the current BitTorrent system. In particular, BitTorrent's rate-based tit-for-tat incentive is instantaneous and seeks to match peers with similar upload capacities, thereby forming bandwidth symmetrical clusters. Thus, due to this technical constraint, a user's download performance is always limited by the download rate of its corresponding cluster, no matter how much data it has contributed to the system over time [93].

In contrast to this, multilateral incentive strategies detaching from time enable that reward can be both instant and deferred. This property aligns well with the highly asymmetric nature of residential broadband links. Specifically, users (i.e, lower provisioned nodes) could contribute for a longer period of time (e.g., when not actively using their desktop PC) and receive reward for those contributions at a later point in time at a possibly much higher download rate [67].

From this discussion, we therefore derive our first important solution requirement.

Requirement 1 (Multilateral Incentive Design). *The solution must be designed for detaching incentives from time, content, and individual peers.*

4.4.2 *Decentralisation*

At first glance, Requirement 1 can easily be achieved by assuming a central authority infrastructure that mints identities, keeps account of user contributions, and pun-

ishes uncooperativeness. However, there are a few practical issues that prevent from deploying a large scale P2P content distribution on top of a central coordinator.

First, our long-term measurements in the BitTorrent system have shown that user population sizes can quickly exceed several millions of online users. For this reason, in practise, a central coordinator would have to withstand extremely high load requirements as every single online user is going to interact with it. Consequently, it is clearly a bottleneck and single-point of failure. Nevertheless, even if the reliability and stability of the coordinating instance can be guaranteed, a central coordinator still remains untrustworthy for many users, because it is usually under the control of a single administrative domain.

Therefore, in favour of scalability, reliability, and openness of the approach, we consider a solution that follows the principles of the P2P paradigm and thus avoids any centralised authority infrastructure to which peers must register or even interact with. Instead, the required functionality must be implemented in a decentralised manner by efficiently leveraging the resources of the participating users.

Requirement 2 (Decentralisation). *The solution must be designed to operate without the involvement of any pre-trusted or centralised authority infrastructure.*

4.4.3 *Robustness*

When implementing a multilateral incentive strategy in a decentralised manner, peers need to exchange messages about third-party behaviour. This unavoidable fact, however, greatly expands the range of possible attacks [90]. In particular, depending on the extend of used third-party information, the incentive scheme may quickly become vulnerable to *Sybil* attacks, *whitewashing*, and *collusion* [26, 34, 74]. For instance, if new identities can be created at minimal costs and this process is not controlled by an authorised infrastructure, any user can create multiple Sybil identities for the purpose of boosting its reputation in the system [26]. Further, even if an attacker is detected in the system, it can always 'whitewash' its standing by creating a new identity, thereby avoiding any punishment associated with its old account [34]. Finally, peers may collude with others to report fake contributions, thereby obtaining services from the community for free, without the need of contributing any [74].

In general, Cheng et al. have shown that multilateral incentive strategies that do not feature a certified user registration (e.g., with the aid of a public-key infrastructure) can never completely defend against these kind of attacks [19]. The best one can hope is thus to reduce the effectiveness of these attacks, leading to the following requirements.

Requirement 3 (Robustness to Impersonation). *The solution must be designed to hinder a malicious peer to adopt the identity of other participants in the system, thereby preventing misbehaviour in the name of the victim.*

Requirement 4 (Robustness to Rational Attacks). *The solution must be designed to provide robust countermeasures against peer collusion, whitewashing and the Sybil attacks such that the benefit from running one of these attacks is limited.*

Apart from the fact that multilateral incentives can be subject to manipulation, peer turnover or the lack of protocol incentives can also cause users to lose their reputation. For instance, since the solution is supposed to be decentralised, peers storing reputation information about other users may suddenly crash or leave the system forever. Further, since upload capacity is a rare resource in online content distribution, selfish users may be tempted to discard protocol messages that are not beneficial to them. From these two important facts, we identify to two further solution requirements.

Requirement 5 (Robustness to Churn). *The solution must be designed to allow peers to reliably store and lookup experiences about third-party behaviour, regardless of whether some nodes suddenly crash or leave forever.*

Requirement 6 (Self-sustainability). *The solution must be designed to incite users for propagating protocol relevant messages.*

4.4.4 *Contribution Fairness*

The performance analysis of the BitTorrent system has revealed that users are highly sensitive to their perceived quality of service. Specifically, more than 45% of the users aborted their download due to poor performance (cf. Section 3.3.2). From this observation, it can therefore be deduced that, when improving availability, any viable solution must also maintain an acceptable level of performance. This is, however, challenging because optimising for overall system utility often requires exploiting the resources of high capacity nodes [31]. These users hold the majority of total upload capacity in P2P content distribution systems and must therefore be kept engaged as long as possible in the distribution process [50]. However, when treated unfairly, it is likely that these nodes will withhold their resources, resulting in a performance collapse on the long-term. This is evidenced by the fact that due to fairness issues users already start to throttle their upload contributions in the BitTorrent system [94]. Therefore, we derive the following important requirement.

Requirement 7 (Contribution Fairness). *The solution must be designed to provide service differentiation that is proportional to the upload contribution of the participants.*

4.5 CONCLUSIONS

This chapter has explored the potential of different abstract solution approaches to overcome the performance and availability issues discovered in the BitTorrent system. Three cross-torrent incentive approaches as well as single-torrent incentive alternatives have been considered and quantitatively analysed through trace-based simulations. Most notably, it was found that:

- *Single-torrent approaches are infeasible.* Any approach that attempts to lengthen the online times of the users upon completing their downloads (e.g., with monetary awards or file encryption) emerges as highly impracticable. The results show that, on average, seeding times would have to be increased by a factor of 10 to gain 99% file availability. In other words, after finishing the download, a user would have to stay online more than 34 hours on average.
- *Bilateral cross-torrent approaches are insufficient.* To achieve long-term availability of content *and* high download performance in the BitTorrent system, any cross-torrent solution based on bilateral incentive strategies is insufficient. The reason for this is that peers lack important trading opportunities because *(i)* they do not have synchronous file interests and *(ii)* most of them (81%) only meet once in the system.
- *A solution requires a multilateral cross-torrent approach.* The trace-based simulations have shown that cross-torrent incentives that are not only detached from time and content, but also from *individual* peers have a great potential to overcome the performance and availability issues in BitTorrent. In such an approach, peers would be able to claim back contributions at any time from any peer, i.e., reciprocation would be indirect.

Through these findings, we were able to reduce the abstract solution space to the class of multilateral incentives strategies that are agnostic to time, content, and individual users. With this important information, we have deduced a set of key requirements that a solution must fulfil to address the seeder promotion problem in the BitTorrent system. These requirements also take into account the research challenges presented in Section 1.3. Consequently, the next step is to design a concrete solution striving to fulfil these requirements.

5 A NEW P2P INCENTIVE MECHANISM – FAIRSWARM.KOM

The previous chapter has revealed a set of key requirements for addressing the seeder promotion problem in the BitTorrent system. This chapter now focuses on the design and implementation of FairSwarm.KOM[1], a new P2P incentive mechanism that is devised to meet these requirements.

5.1 INTRODUCTION

The quantitative analysis of the abstract solution space has provided a basic understanding of the best general approach to address the seeder unavailability problem. Through this we derived a set of key requirements that a solution must fulfil to overcome the discovered performance and availability issues. This chapter details our concrete solution approach, named *FairSwarm.KOM*, capable of meeting these requirements.

FairSwarm.KOM's design is driven by properties such as no centralised trust, robustness (e.g., against churn and strategic manipulations), and self-sustainability. This means, it has been taken into account that all protocol aspects are clearly motivated and each peer takes its role in the system voluntarily, as everything happens in its own interest. Otherwise, due to the assumed selfishness of the users, the protocol would be difficult to implement in practise.

In general, FairSwarm.KOM is a *generic* incentive solution that can be adapted to a variety of P2P content distribution systems, without affecting their key functionality. In the following, however, we opt for presenting details on how this protocol can be incorporated in BitTorrent, because it represents the current de-facto standard for scalable content distribution. Specifically, BitTorrent is nowadays used by millions of users world-wide that are transferring terabytes of data per day. Hence, a solution for BitTorrent is seen as the most urgent.

At first, we give a conceptual overview of FairSwarm.KOM's principle elements and their key functionality. Subsequently, we focus on each of these elements in more detail, while important design decisions are justified. After this, we delineate how the approach can be implemented in the BitTorrent system. Finally, we revisit the solution requirements introduced in Section 4.4 to show how they have been addressed by our design.

5.2 CONCEPTUAL OVERVIEW

FairSwarm.KOM is a pairwise receipt-based indirect reciprocation mechanism that allows peers to evaluate the long-term behaviour of other nodes by using information that goes beyond direct observations and local histories. To achieve this, peers have permanent identifiers and persistently document their bilateral interaction pattern with other nodes using digital receipts, so-called *coupons*. These coupons are then propagated in the network to enrich the knowledge base of other peers. Accordingly, a peer's local view of the contribution network is solely constructed from the coupons gossiped by the other nodes.

1 In the originally submitted version of the thesis the name FairTorrent was used for the concept represented by FairSwarm.KOM. It was subsequently discovered that the notion FairTorrent has been also used by Sherman et al. [109] for an unrelated concept. In order to avoid confusion, we therefore have decided to use the notion FairSwarm.KOM throughout. This does, however, neither change the original idea, underlying concept or affect the context of the work. It is a simple name change.

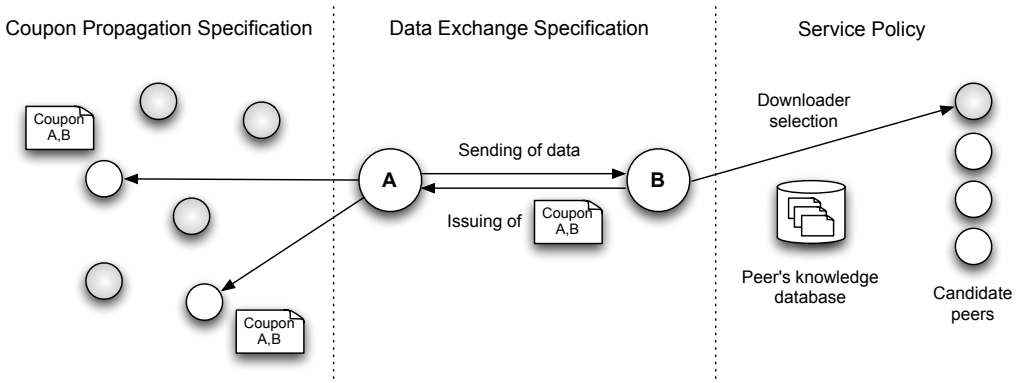


Figure 35: Overview of principle elements of FairSwarm.KOM.

The sole requirement to participate in FairSwarm.KOM is that each peer possesses a cryptographic key pair, consisting of a 1024-bit long public and private key[2]. This is necessary to enable long-term user identification and authentication. To this end, users can freely create their own public/private key pairs. Thus, FairSwarm.KOM does not require any complex public-key infrastructure to which users must register, nor do users have to pay for participation.

To enable indirect reciprocation in a fully distributed manner, FairSwarm.KOM consists of four principle elements: *(i)* coupons, *(ii)* a coupon propagation specification, *(iii)* a data exchange specification, and *(iv)* a given service policy. Figure 35 gives an overview of these elements as well as their interaction. The key functionality of these elements can be briefly summarised as follows:

- *Coupons.* In FairSwarm.KOM, *coupons* are bidirectional transaction receipts, attesting the long-term exchange of data between two peers. Each coupon is mutually signed by both interaction partners and thus only contains information to which both parties agree. Through these signatures, it is also prevented that third-parties can modify coupons at a later date, i.e., to gain an advantage by undermining the contributions of other users in the system.

- *Coupon Propagation Specification.* In order to claim back contributions, peers can submit coupons to third-parties by following FairSwarm.KOM's coupon propagation specification. Instead of simply flooding coupons in the network, FairSwarm.KOM exploits the *long-term* interaction pattern of the users and thus limits the propagation of coupons to at most *one hop* in the overlay network. This restriction fosters scalability and provides increased robustness (i.e., to untruthful peer behaviour and peer turnover). In addition to this, the specification also provides mechanisms to meet the trade-off between communication costs and information up-to-dateness. For instance, it is not always necessary to inform overlay neighbours about every single transaction occurred.

- *Data Exchange Specification.* FairSwarm.KOM's data exchange specification provides a particular safety concept, allowing two users to exchange digital items in a *fair* way. That is, when running this specification, it is guaranteed that both parties either obtain the item they desire, or neither party does. Providing such functionality is crucial for a receipt-based indirect reciprocation scheme, because once a service provider has properly delivered the requested data, it is convinced that it will receive a corresponding coupon for its service provisioning. On the other hand, once a service consumer has attested the receipt of the requested data, it can be sure to actually receive this data.

2 To date, this key length can be considered as secure enough to perform standard cryptographic operations such as digital signatures on messages.

- *Service Policy.* The service policy is responsible for performing peer selection among a set of interested candidate peers. Depending on the instantiation of such a policy, different objectives can be pursued. For instance, a service policy can be adjusted very conservatively to provide robustness against strategic attacks, optimised for total system throughput or configured to foster fairness. Finally, a mix of all of these aspects is possible, as exemplified by our default service policy. Thus, FairSwarm.KOM decouples the mechanism prescribing how to use reputation information (the Service Policy) from the mechanism providing this information (the Coupon Propagation Specification). This design choice is intentional, because it provides system designers with the flexibility to evolve their own policy according to a large variety of criteria.

Next, we focus on each of these principle elements in more detail, both to delineate its concrete design and to further explain the rationale behind our design decisions.

5.3 COUPONS

In contrast to prior approaches that issue digital receipts for every single transaction (e.g., [69, 80]), FairSwarm.KOM follows a novel cumulative approach that stores the long-term interaction pattern between two peers in a single digital receipt, called a *coupon*. As such, each coupon is a data container providing information about the total amount of data transferred between two peers (in both directions). Since this information can change over time (e.g., when peers interact with each other), each coupon has a version number that can be updated by both coupon producers.

The approach to stepwise document the bilateral interaction pattern between two peers is advantageous because of the following reasons. First, by transmitting only the coupon's latest version, a third-party is immediately informed about the complete transaction history of both peers. Moreover, it can directly infer how much data each party has provided *and* consumed. In contrast to this, when using per-transaction receipts, selfish peers are tempted to suppress the existence of receipts attesting their service consumption in the network. Thus, it is in the responsibility of other users to complete this picture (i.e., to transmit the missing receipts), which often complicates the approach. Finally, especially in multi-source downloading, a data transaction relates to the transfer of a single chunk. Therefore, transferring only one coupon that summarises a large series of transactions is much more efficient (in terms of network traffic and overhead) than transmitting a multitude of receipts for every single transaction that has occurred.

As illustrated in Table 6, the concrete design of each coupon can be decomposed into six data fields and two optional signatures. Next, we describe these data fields in more detail and explain the intuition behind the signatures.

5.3.1 *Coupon Data Fields*

In each coupon, the first two data fields are reserved for the unique identifiers (IDs) of the corresponding peers to which the coupon relates. To this end, we say that peer A and B are *producers* or *generators*[3] of coupon $\zeta_{A,B}$. The IDs of both coupon producers are computed by applying a collision-free hash function $h(\cdot)$ to the public key K_{pub} of each producer: $ID = h(K_{pub})$. For this purpose, FairSwarm.KOM uses the SHA-1 hash function generating a 160 bit long identifier [125]. Furthermore, as depicted in Table 6, the order of the producers' identities can vary in the first two fields, depending on who was the service provider and consumer in the most recent data transaction.

The third data field of the coupon is reserved for the ID of a so-called *mediator*. A mediator is only involved in the data exchange process if one party tries to cheat the

3 We use both terms interchangeable in the remainder of this chapter.

Data field	Size	Description
ID_P	20 bytes	Identity of the peer that was service provider in the most recent transaction
ID_C	20 bytes	Identity of the peer that was service consumer in the most recent transaction
ID_M	20 bytes	Identity of the peer that was mediator in the most recent transaction
Data volume $P \longrightarrow C$	4 bytes	Total amount of data transferred from P to C
Data volume $C \longrightarrow P$	4 bytes	Total amount of data transferred from C to P
Version number	4 bytes	The version number of the coupon
Signature of provider	46 bytes	Cryptographic signature of the provider on the six data fields
Signature of consumer	46 bytes	Cryptographic signature of the consumer on the six data fields + signature of provider

Table 6: Each coupon is composed of six data fields and two signatures of the corresponding producers.

other. For instance, if peer B has obtained data from A, and yet denies having received this data, the mediator will get contacted by peer A to resolve this conflict. This issue is discussed in detail in Section 5.5.

To document the long-term transaction history of peer pair (A,B), the fourth and fifth data fields store information about the total amount of data sent from peer A to B, and vice versa. Upon the successful completion of a data transaction, these two data fields are updated according to transactional parameters (i.e., who was the provider and the consumer, and how much data has been transferred). In addition to this, the coupon's version number is incremented (sixth data field). Importantly, since the most recent coupon version cumulates information over all previous interactions of a given peer pair, it is sufficient to possess or distribute only this version in the network. Similarly, once a peer receives a coupon whose version number is higher than the one it already possesses, it simply discards the older coupon.

5.3.2 *Coupon Signatures*

The last two fields of each coupon are reserved for the signatures of both parties. In general, any digital signature scheme can be used for this purpose, e.g., the RSA algorithm or Digital Signature Algorithm (DSA) [105, 89]. It is important to note that both signatures are generated successively, beginning with the signature of the data provider. We present details about this process, as well as the rationale behind it, in FairSwarm.KOM's data exchange specification (see Section 5.5).

We say that a coupon is *valid* if and only if it contains the signature of both the producer *and* the consumer. If this condition is not fulfilled, a coupon is useless in the system. That is, it is simply ignored by other users, as a fraud attempt seems likely. If a coupon, however, contains these two signatures, third-parties can assume that the coupon was generated in mutual agreement of both parties. In particular, since $h(\cdot)$ is collision-free, it is hard to find a second public key K'_{pub} that hashes the same result $h(K'_{pub}) = h(K_{pub})$. Thus, when a third-party can successfully verify both signatures, it is guaranteed that the coupon truly originates from the producers to which the coupon relates, providing *authentication*.

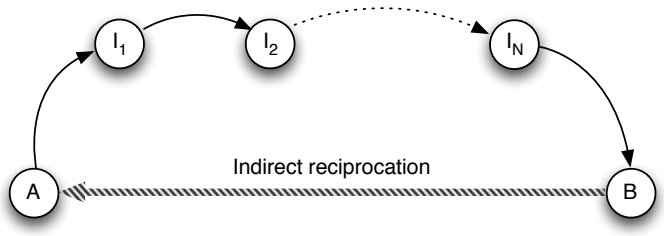


Figure 36: Indirect reciprocation via multiple intermediary nodes.

5.4 COUPON PROPAGATION SPECIFICATION

This section delineates FairSwarm.KOM's *coupon propagation specification* that allows peers to exchange information about third-party behaviour in the network. We begin by discussing a major design decision taken; the propagation of coupons to at most one-hop in the overlay neighbour network. Subsequently, we describe how the resulting indirect reciprocation infrastructure is bootstrapped and maintained in FairSwarm.KOM.

5.4.1 *Selection of Indirection Level*

Our solution space analysis has revealed that direct reciprocity is insufficient to overcome certain performance and availability issues, because *(i)* most peers only interact with a few others and *(ii)* users do not always simultaneously require each other's content. Instead, it has been shown that a solution requires a multilateral strategy that is agnostic to time, content, and individual users (cf. Requirement 1 of Section 4.4).

Indirect reciprocity has the potential to fulfil this solution requirement. In particular, it allows users to contribute to one peer yet receive reciprocation from another peer at a later point in time. More importantly, these contributions do not have to take place in the same overlay swarm. For instance, as illustrated in Figure 36, peer A could initially send data to peer I_1, peer I_1 could in turn send data to I_2 in another torrent, and so on. Finally, when peer A and B encounter each other, peer B recognises A's cooperativeness along the contribution path and indirect reciprocation can take place. This approach improves incentives because it increases the amount of users from which peer A can receive reciprocation from (indirectly and in the future).

Importantly, however, as peers only have a local view of the system, peer B can only recognise the contribution path from $A \rightarrow I_1 ... \rightarrow B$, if it is informed about *all* the interactions that have been occurred between the intermediary nodes (across time and torrents). To achieve this, peer B must share information about third-party behaviour with its direct interaction partner I_N. This peer, in turn, must forward experiences of its previous or current interaction partners (e.g., peer I_{N-1}) to peer B, and so on. Intuitively, with each additional *level of indirection* used, peer B's knowledge base of the contribution network grows exponentially, which clearly increases the probability of detecting a contribution chain from A to B, enabling indirect reciprocation.

Ideally, it is desirable to have as much information as possible for decision making (i.e., to select the highest level of indirection possible). However, care must be taken as there is an inherent trade-off between different properties. For instance, when moving to higher levels of indirection the communication overhead grows exponentially [131]. Accordingly, control traffic may quickly become a dominating factor and peers might spend more time sending status messages than transferring user data. In addition to this, the higher the level of indirection used, the more vulnerable is the approach to untruthful peer behaviour. Specifically, as long as the peers' knowledge base is limited to one level of indirection, peers still can locally judge about the trustworthiness of

information of intermediate nodes. This is because the peers only accept experiences about third-party behaviour from overlay neighbours they have directly met. For any higher level, however, peers must believe in the recommendations of the neighbours' neighbours, and so on. Thus, transitive trust mechanisms must be applied that greatly expand the range of available attacks to strategising or even malicious users.

On the other hand, if the level of indirection is chosen too low, peers might have an insufficient knowledge base to take proper decisions regarding whom to reward for previous contributions. This leads back to the problem that we have with direct reciprocity in currently deployed P2P content distribution systems. Therefore, when attempting to find a good balance between this issue and the different trade-offs mentioned above, the most important question is: What level of indirection is sufficient to promote *long-term* incentives for users? Or, to put it another way, given a random peer pair in the interaction graph: How many intermediary nodes are *at least* required to find contribution cycles between these nodes?

To ascertain this, we have utilised our long-term measurement data and performed an analysis of the interaction pattern of the nodes, across torrents and time (see Appendix A.4). Through this, we find that the current BitTorrent system exhibits small world patterns. That is, peers cluster together based on their social interests, and, more importantly, the majority of peers (74%) are indirectly connected to each other via a small minority of 'power' users accounting for more than half of the total system demand. To this end, it can be shown that, when exploiting this one-hop relationship, more than 96% of these users could claim back contribution as seeders immediately in their next file download.

As one of our design principles is to keep things as simple as possible, we exploit this observed small world pattern and thus opt for limiting the propagation of coupons to one-hop in the overlay network, in favour of scalability and robustness. While this design decision has the clear consequence that peers have only a very restricted view onto the contribution network, we believe it is still a good trade-off between the different properties. Furthermore, in Chapter 6, we also quantify what impact the information loss has with regard to fairness and performance, compared to schemes using global knowledge of the network (shared history).

5.4.2 *Bootstrapping of Indirect Reciprocation*

To bootstrap one-hop indirect reciprocation, peers must learn about common interaction partners when encountering each other. To illustrate this, consider the example of Figure 37. Here, peer A sends data to peer I in swarm 1, while after some time, peer I sends data to peer B in swarm 2. When peer A and B meet in swarm 3, peer B must recognise that A has indirectly contributed to B via intermediate node I. Importantly, this must be possible without the requirement that peer I is currently online in the system as this is often not the case, as described below. Thus, to meet this requirement, each party must simultaneously perform the following steps during connection establishment:

1. First, peer A produces a set S_A, consisting of the identifiers of A's prior interaction partners[4]. This set is then sent together with A's public key to peer B, using a `Bootstrap Msg` (cf. Table 7). Note that peer A's public key must only be transferred, when both peers encounter each other for the first time. Otherwise, A's public key will already be contained in peer B's local database.

2. Upon receiving the `Bootstrap Msg`, peer B generates set S_B consisting of the identifiers of B's previous interaction partners. Subsequently, it computes the intersection of the sets S_A and S_B, denoted by $S_I = S_A \cap S_B$. As such, S_I contains the IDs of all intermediate nodes that both peers have in common.

4 Interaction partners of peer A are all those nodes with which A has a common coupon.

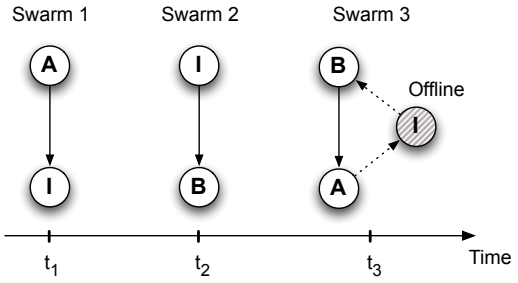


Figure 37: One-hop indirect reciprocation across time and torrents.

Message Type	Data Fields
`Bootstrap Msg`	– Public key of peer A – List of identifiers of A's prior interaction partners
`Bootstrap Reply Msg`	– Public key of peer B – Coupons of common interaction partners of peer A and B – Most favourable coupons of peer B + necessary public keys
`New Peer Msg`	– Identifier of the new interaction partner
`Coupon Msg (A,B)`	– Most recent version of coupon $\zeta_{A,B}$

Table 7: Table of messages.

3. For each intermediate node $I \in S_I$, peer B fetches the most recent coupon from its local database and sends these coupons to peer A using a `Bootstrap Reply Msg`[5]. Since peer A knows all those intermediate nodes from prior interactions, the transferred coupons can be used to evaluate peer B's contributions in the past. This is essentially the main source of information to enable one-hop indirect reciprocation.

4. Additionally, by using the same message, peer B selects a list of coupons that are most favourable to it and sends these as well (including the necessary public keys). We say that a coupon is favourable for a peer if it attests a positive account balance for the node (i.e., the difference between the amount of provided and consumed data is greater than zero). Although peer A will not use these coupons to evaluate peer B, they are important to detect fraud attempts (see below).

5. Last, upon receiving the `Bootstrap Reply Msg`, peer A performs the following two checks to verify peer B's honesty:
   a) *Fraudulent concealment of intermediaries.* The first check tests whether there is an intermediary node I that both peers have in common, but a selfish peer B tries to conceal (i.e., coupon $\zeta_{I,B}$ attests a negative account balance for B). To test this, peer A only needs to verify if it already has a coupon $\zeta_{I,B}$ in its local database that is not contained in the `Bootstrap Reply Msg` of peer B. If this is true, peer A immediately bans peer B.
   b) *Replay attacks (withholding of newer coupon versions):* The second check, on the other hand, tests whether peer B has actually provided the most recent coupon version of all coupons contained in the `Bootstrap Reply Msg`. This

5 Note that the public keys of the intermediate nodes must not be transferred, as those are already stored in A's local database due to prior interactions.

is necessary because a selfish peer B could perform a replay attack by sending an out-dated, but most beneficial coupon version.

It is important to note that the key to successfully prevent *(i)* the concealment of intermediaries and *(ii)* replay attacks with older coupon versions, lies in the proactive gossiping of a node's most favourable coupons (step 4). As mentioned before, although peers do not directly benefit from this coupon gossiping, it forces rational/selfish nodes to honestly report common intermediate nodes and to send only a coupon's most recent version. In particular, a peer's most favourable coupons are exactly those that a dishonest node primarily seeks to suppress, because they most impair the node's standing in the system. However, particularly because of this coupon gossiping, when encountering a new peer, a dishonest node never knows which coupons the counterpart already possesses. This will highly restrict one of the above attacks, as a peer has always to weigh-off between boosting its own account balance and not receiving any data at all.

Finally, as coupons are tamper-proof and summarise the entire transaction history between two nodes, peers can propagate those receipts in the system, even if their producers have already left the system. This approach is intentional, because it circumvents the need for requesting intermediaries to attest contributions (i.e., as performed in recent works such as [94, 15]). We believe that this approach is the only way to bootstrap one-hop indirect reciprocation across time. Specifically, our long-term trace data of the BitTorrent system reveals that the probability that an intermediary node is online when two peers meet in the system, is only 29%. Thus, when relying on online intermediaries, the effectiveness of approaches is significantly impaired, because in most cases peers cannot approve their contributions.

5.4.3 *Maintaining Indirect Reciprocation*

During connection establishment, peers learn about previous interaction partners that both peers share in common. This set of intermediate nodes, however, remains static over the uptime of the nodes. Nonetheless, peers learn about new nodes when downloading content and can thus also form new one-hop relationships. Consequently, a mechanism is needed to detect such cycles instantly, as soon as they arise.

To accomplish this, our devised mechanism works as follows: Whenever a peer A learns about a new interaction partner N, it sends a `New Peer Msg` to all overlay neighbours (cf. Table 7)[6]. This message only consists of the identity of peer N and is thus very lightweight. Each recipient R, on the other hand, checks whether it possesses a coupon with this node. If this is true, sender A and recipient R mutually exchange their corresponding coupons with N, using a dedicated `Coupon Msg`.

Figure 38 illustrates this algorithm using three overlay nodes A, B, and I. In this example, we only consider the signalling traffic of peer A. The process remains the same, however, for the other two peers. Assume now that all peers are already connected to each other. Depending on the interaction pattern of the peers, the following messages are exchanged between nodes:

1. Peer pair (A, B) is the first to interact with each other. Since this is also the first interaction between the two nodes, a new coupon is generated. Subsequently, peer A sends a `New Peer Msg` to peer I, containing the identity of B. As peer I has never interacted with node B before, it simply discards the message.

2. At a later date, peer A also starts to exchange data with peer I and informs B about its new interaction partner using a `New Peer Msg`. Similarly to situation (1.), peer B discards this message, because it has no transaction history with I.

6 Recall that an interaction partner N is a peer with which a node A has exchanged at least one data unit. Thus, a valid coupon $\zeta_{A,N}$ exists in the database of both peers.

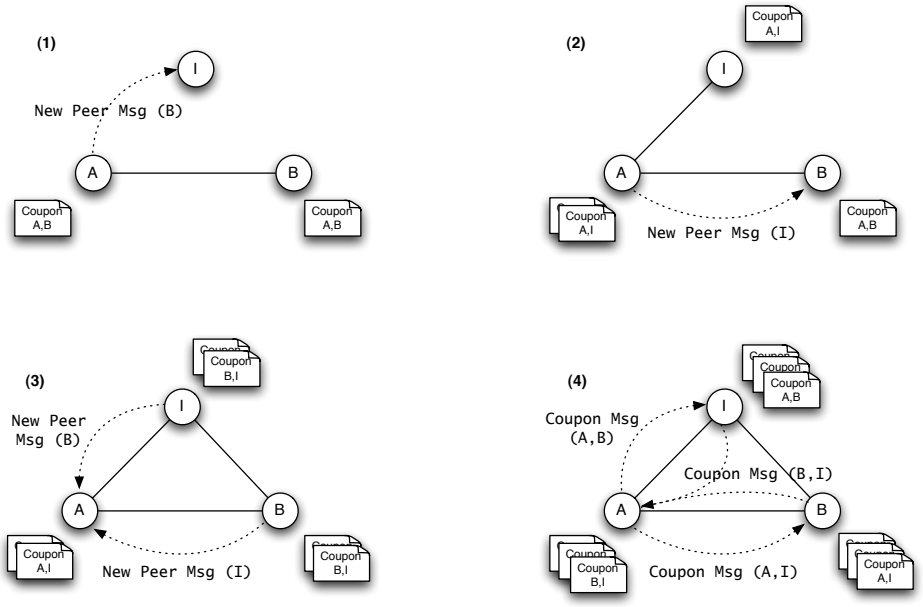


Figure 38: Formation of new triangular relationships to enable instant one-hop indirect reciprocation (peer A's point of view).

3. After this, peer B and I begin to interact with each other. As a result of our algorithm, peer A receives from both peers I and B a `New Peer Msg`.

4. Upon receiving these messages, peer A fetches the corresponding coupons from its local database, and sends $\zeta_{A,B}$ to peer I and $\zeta_{A,I}$ to peer B, respectively. These nodes answer with coupon $\zeta_{I,B}$ and a new triangular relationship is formed enabling one-hop indirect reciprocation.

Note that the above specification is self-sustaining, because one of both interaction partners has an inherent incentive to report the exchange of data with the newly encountered node. In particular, in situation (3.) of Figure 38, the service provider of the data transaction between peer I and B is highly interested in making this service provisioning publicly available. Thus, in the worst case, either I or B will send the corresponding `New Peer Msg`, which guarantees that A learns about $\zeta_{I,B}$ in step (4.).

5.4.4 *Managing Coupon Versions*

To inform overlay neighbours about the generation of a new coupon version, FairSwarm.KOM follows an incentive-compatible *push*-approach. In particular, each time a peer A generates a coupon $\zeta_{A,B}$ together with another peer B, it compares its current account balance to node B with the balance of the coupon version that has been previously submitted to other users. We define the account balance by the difference between the amount of provided and consumed data. If the difference between the two balances *(i)* is positive (i.e., the new coupon improves the peer's standing) and *(ii)* exceeds a given percentage threshold (e.g., $>10\%$), the new coupon is sent to common interaction partners of both peers using a dedicated `Coupon Msg`.

Restriction *(i)* prevents both parties from having to perform the redundant role of informing their common interaction partners about the generation of new coupon versions, which would effectively double the overhead traffic. Furthermore, giving the update responsibility to the peer with the positive difference between the two account balances is also intentional, because this peer has an inherent incentive to submit the

coupon (i.e., it improves its chances when competing for download slots from other peers).

Restriction *(ii)*, the percentage update threshold, tackles the trade-off between a peer's profit from submitting a coupon and the communication costs arising. In particular, the introduction of a percentage threshold results in frequent coupon updates when a peer's reputation is low (i.e., its account balance is around zero), which is desirable. On the other hand, the more a peer's account surplus/deficit increases, the more transactions are needed to be performed until a new coupon version is submitted. This is also desirable as peers must not be informed about the increase/decrease of a peer's account balance until the surplus/deficit amounts to a relatively high value, i.e., tens of megabytes. In these ranges, minor information updates would probably not influence the decisions taken by other nodes. Instead, it is only important to communicate major changes, thereby saving signalling traffic.

5.5 DATA EXCHANGE SPECIFICATION

The success of FairSwarm.KOM crucially depends on the fact that peers obtain coupons for their data provisioning, which can then be disseminated in the system. Consequently, it is of particular importance that the issuing of these transaction receipts is closely tied to the actual delivery of data. To this end, a sophisticated mechanism is needed that allows two potentially mistrusting parties to exchange these items in a *fair* way. This means, either the service provider obtains a valid coupon for its data delivery and the service consumer actually receives the data it has requested, *or*, both peers receive nothing. In this section, we present FairSwarm.KOM's *data exchange specification*, capable of fulfilling this need.

5.5.1 *Requirements of Specification*

While the simultaneous exchange of items can easily be performed in real life (e.g., face-to-face on a counter), the process becomes much more challenging in asynchronous systems (i.e, the Internet). In particular, Even and Yacobi [29] have shown that, in these kinds of systems, a fair exchange of two digital items is impossible to achieve in a *deterministic* two-party protocol. Instead, exchange fairness always requires the involvement of an intermediate node, also deferred to as *mediator*.

This observation has a major impact on the design of our data exchange specification. In particular, it says that, without the involvement of a third-party, it is impossible to guarantee that *(i)* a service provider actually receives a valid coupon for its data provisioning, while *(ii)* the service consumer truly obtains the data it has requested and attested. To see this, consider the simple state model of Figure 39. In the initial fair state (beginning of a transaction cycle), each party possesses an item in which the other party is interested in. That is, the provider owns the data while the consumer has the valid coupon that later on attests the receipt of this data. Starting from this state, the goal is to achieve the desirable final fair state in which both parties have obtained the other's item. However, as the communication over the Internet is always asymmetric and non-simultaneous, both peers have to go through an unfair intermediate state. Accordingly, one of both peers receives its item first; a circumstance that can be systematically exploited by a dishonest node. Specifically, a malicious node could misbehave as follows, depending on the role it is currently performing:

- Upon receiving the requested data, a dishonest consumer may deny having received this data. For instance, it refuses to send a valid coupon attesting the provider's data provisioning.

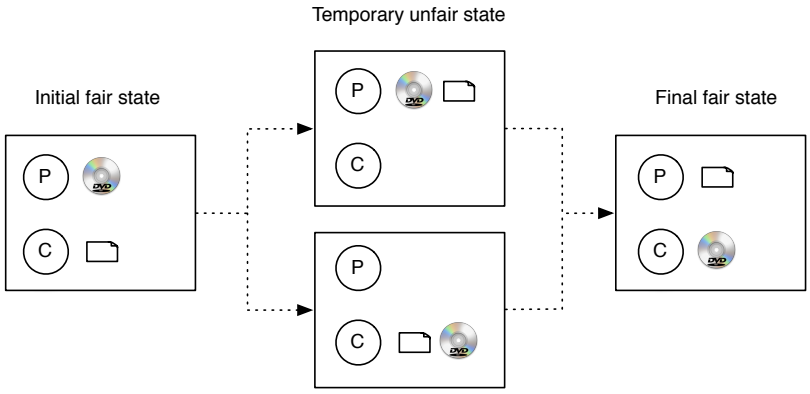


Figure 39: Exchange process of digital items between two parties in a non-simultaneous and asymmetric computer network. P and C denote the service provider and consumer that are initially possessing a data object and coupon, that is to be exchanged, respectively.

- Upon receiving a valid coupon, a dishonest provider may not upload any data and yet claim to have delivered this data. For instance, it could submit the coupon to third-parties.
- Upon receiving a valid coupon, a dishonest provider may upload garbage data, either on purpose or due to a communication failure and yet claim credit for this data provisioning, e.g., again by submitting the coupon to third-parties.

It is intuitive to see that as soon as only one of the above attacks is possible, the information provided by the coupon propagation specification would be unreliable and useless. Consequently, any realistic approach to address these attacks must rely on a mediator node. Due to scalability reasons, however, the mediator's involvement should be reduced as much as possible. For instance, a straightforward but undesirable solution is that both parties first submit their items to the mediator. Subsequently, the mediator verifies the correctness of both items (i.e., if the coupon is properly prepared and the data item is valid) and forwards them to the corresponding recipients. In such a solution, however, a mediator node must act as data relay, and is, thus, likely to become a bottleneck for a large-scale P2P content distribution system.

To remedy this, our purpose must be to design a *data exchange specification*, capable of fulfilling the following requirements:

- *Exchange Fairness.* Any possible execution of the specification ends with one of the following two results: *(i)* either the service provider has the corresponding coupon for its data provisioning, and, at the same time, the consumer has received the requested data, *or*, *(ii)* both peers go away empty-handed. Thus, the mechanism is robust against all three types of attacks mentioned above.
- *Data Integrity.* A service provider can only obtain a coupon for data that fulfils a predetermined structure to which both parties have mutually agreed. This prevents peers from uploading garbage data.
- *On-demand Mediator.* If both peers follow the protocol specification, the exchange of data as well as the issuing and sending of the corresponding coupon can be settled *without* the involvement of a mediator. A mediator is only used as soon as one party is misbehaving, i.e., it tries to cheat the other.
- *Timeliness.* The provider as well as the consumer can terminate the protocol specification at any time. Importantly, there is no need for the cooperation of the other node, while exchange fairness still remains guaranteed. This is of particular importance as peers in P2P systems can disconnect arbitrarily at any time without notice.

Within the remainder of this section, we first give an conceptual overview of the devised specification that fulfils these requirements. Following this, we present details about the protocols belonging to this specification. We then conclude with a formal confirmation of the properties achieved by our design.

5.5.2 *Specification Overview*

The data exchange specification of FairSwarm.KOM consists of an exchange protocol, and three additional sub-protocols, i.e., the abort protocol, the recovery protocol and the complaint protocol. The exchange protocol is the main protocol that needs to be run for the exchange of data between two peers. As long as both peers follow this protocol and no communication error occurs, exchange fairness is guaranteed without the involvement of a mediator. That is, after the protocol execution, the provider obtains a valid coupon for its service provisioning and the consumer receives the requested data. We consider this situation as the normal case, as deviating from the protocol is not beneficial. If a peer still does, one of the remaining three sub-protocols is run to achieve exchange fairness with the help of a mediator. Depending on the concrete circumstance, the mediator issues a receipt to declare a coupon as invalid or helps to recover the requested data.

The basic idea of the *exchange protocol* can be briefly explained as follows (cf. Figure 40). Upon receiving a data request from a consumer, the provider decides whether it wants to deliver the requested data. If so, it encrypts the data by using symmetric cryptography and sends this to the consumer. Additionally, the provider generates a new coupon that updates the coupon's most recent version according to the parameters of this transaction (i.e., it sets who is the provider and consumer, and updates the total amount of data transferred from the provider to the consumer). This coupon is then signed and also sent to the consumer. Upon receiving this message flow, the consumer tests the correctness of the data fields of the new coupon and verifies the provider's signature on these. Subsequently, the consumer signs both the new coupon and the provider's signature, and sends this irrefutable *evidence of receipt* back to the provider[7]. The provider, on the other hand, verifies the consumer's signature and ultimately releases the decryption key so that the consumer can decrypt its data. At this time, the protocol ends and both peers have obtained their desired items.

Since the protocol is intended to operate in open platform systems and the communication over the Internet is unreliable, strict adherence to the above protocol specification cannot be achieved in all situations. Instead, it is expected that there are always a few peers that deviate from the above exchange specification, e.g., either on purpose or unintentionally due to communication errors. To handle these abnormal situations, FairSwarm.KOM provides three different sub-protocols (cf. Figure 40):

- *Abort protocol:* This protocol is run by the provider when the consumer fails to send the evidence of receipt (the consumer's signature) in time. This cancels the transaction and the consumer loses its last chance to get the key for decrypting its data.
- *Recovery protocol:* When the consumer provides its signature, but does not receive the key to decrypt its data in time, it may execute the recovery protocol to still achieve exchange fairness. This reconstructs the missing key with the aid of the mediator, allowing the consumer to proceed as normal.
- *Complaint protocol:* This protocol is responsible for resolving situations in which the consumer was able to decrypt the data, but it turns out that the data is garbage. Depending on the evidence provided by the consumer, the mediator

7 Note that only with the consumer's signature, the new coupon becomes valid and can be submitted to third-parties in the system (see Section 5.3).

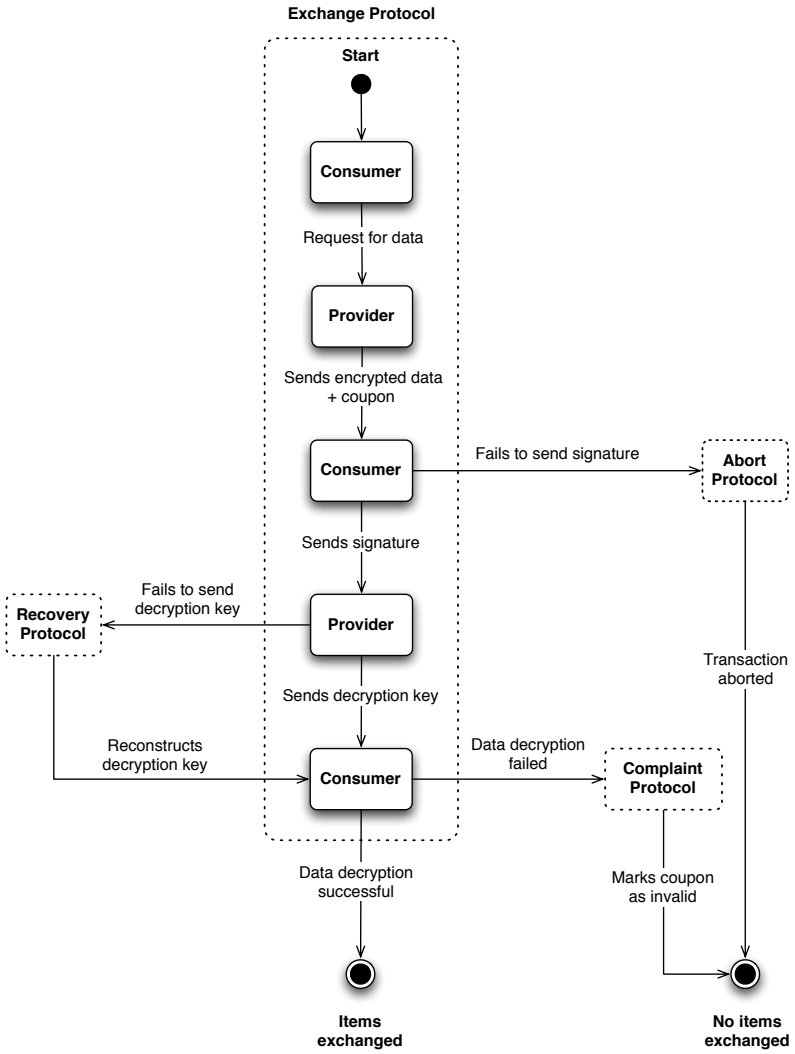


Figure 40: Overview of data exchange specification with its corresponding protocols.

can then declare the corresponding coupon as invalid. Also, it can issue a receipt that marks the identity of the provider as cheater.

In the following, we delineate the exchange protocol and the three different sub-protocols in more detail. For the sake of clarity, however, we first introduce important notations that are necessary for the understanding of the different protocols.

5.5.3 *Protocol Notations*

In each transaction, there are at most three parties are involved, namely the data provider, the data consumer, and a mediator. As described in Section 5.3, the unique identifiers of these nodes are obtained by using a collision-free hash function $h(\cdot)$ that takes a user's public key as the input. We abbreviate the IDs of the above mentioned parties as P, C, and M, respectively. Finally, when we say P, C, or M, we also refer to the corresponding users.

To describe the different protocol steps in detail, we use the notation $< event >$: $< description >$. In this regard, $< event >$ presents an action that can either be the sending of a message from peer X to Y, denoted by $X \longrightarrow Y$, or a local computation, marked by the node's identity that is performing this action[8]. The label

8 Note that peer X and Y are only placeholders for the IDs of the different parties $X, Y \in \{P, C, M\}$.

Actors and Items	
P, C, M	Unique identities of the data provider, data consumer, and the mediator chosen for a given transaction L
O	Data object that is delivered from P to C
O_K	Ciphertext of O, symmetrically encrypted with K
K	The secret session key used to encrypt/decrypt data object O
K_M	Ciphertext of (L, K), using an asymmetric encryption algorithm under the public key of M
$\zeta_{P,C}$	The latest coupon version of P and C, updated according to the transactional parameters of L
L	Unique label to identify a given transaction between P and C, with or without the involvement of M
Cryptography	
$E_X(\cdot), D_X(\cdot)$	Asymmetric encryption and decryption algorithm using the public/private key of users X
$e_K(\cdot), d_K(\cdot)$	Symmetric encryption and decryption algorithm with the secret key K
$Sig_X(\cdot)$	Signature scheme that enables peer X to sign messages or other data objects
$h(\cdot)$	A collision-free hash function
Mediator Receipts	
$REV_{\zeta_{P,C}}$	Informs the recipient that coupon $\zeta_{P,C}$ is revoked.
FA_X	Irrefutable receipt that peer X has tried to cheat the system.

Table 8: Table of notations.

$<$ description $>$, on the other hand, either details the content of the message or gives information about the computation and its conditions.

Furthermore, we use a couple of very basic cryptographic mechanisms that are abbreviated and defined as follows. Let $(e_K(\cdot), d_K(\cdot))$ be a pair of secure symmetric encryption and decryption algorithms that operate based on a secret session key K. For instance, the Blowfish-CBC with a 128 bit key and a 128 bit initialisation vector fulfils this requirement [108]. Asymmetric cryptography, on the other hand, is labelled by capital letters. To this end, $(E_X(\cdot), D_X(\cdot))$ denotes the asymmetric encryption and decryption algorithms using peer X's public and private key, respectively. Finally, $Sig_X(\cdot)$ defines a signature scheme that allows peer X to sign messages or other data objects. As usual, this signature is publicly verifiable by using the public key of peer X. A standard (secure) asymmetric scheme that allows for both purposes is the RSA algorithm [105].

To identify the protocol instance relating to a transaction between P and C, we use a unique *label* L. This label is defined by

$$L = h\left(P, C, M, h(O), h(O_K), h(K), h(\zeta_{P,C})\right),$$

and clearly indicates that P is going to transmit the data object O to C, with or without the help of a mediator M. Note that L contains a hash of the data object that is to be transmitted, a hash of the already updated data fields of coupon $\zeta_{P,C}$, a hash of the secret session key K and a hash of the ciphertext of O that is encrypted with K: $O_K = e_K(O)$. The data contained in L is important to prevent peers from uploading garbage data and/or release an invalid secret key K and yet claim credit

for this. In particular, as shown later on, once a data provider signs (L, K_M) where $K_M = E_M(L, K)$, a mediator can verify at any later date whether P has cheated on C.

Finally, during the execution of the data exchange specification, the mediator has the opportunity to issue a receipt $REV_{\zeta_{P,C}} = Sig_M(\text{'revocation'}, \zeta_{P,C})$ that informs any recipient about the revocation of a particular version of coupon $\zeta_{P,C}$. Further, the mediator can also issue the receipt $FA_X = Sig_M(\text{'fraud attempt'}, X)$ that acts as irrefutable evidence that party X has tried to cheat another peer during the data exchange process. Accordingly, both receipts can be submitted to third-parties that were not directly involved in the transaction.

Even though it is intuitive, it should be still noted that the revocation receipt only relates to the version of $\zeta_{P,C}$, that is well-defined by $REV_{\zeta_{P,C}}$. Earlier versions of this coupon thus remain still valid. For the ease of understanding, Table 8 gives an overview about all introduced notations.

5.5.4 *Exchange Protocol*

A prerequisite for the exchange protocol is that both peers have exchanged their public keys, before the actual protocol starts. This is usually performed during connection establishment. Furthermore, both peers must have agreed to an *independent* mediator M and possess its public key[9]. This is necessary to prevent collusion between a dishonest party and the potential mediator. Unless a problem arises, however, peer M will not be informed of its role.

The exchange protocol begins when the consumer issues a data request for a given object O, as shown in Table 9. Importantly, to uniquely identify data objects in the system, this request message must contain the hash of object O. Assume now provider P is interested in serving this request, but only with the guarantee that it receives a valid coupon for the delivery of object O. Consequently, P fetches the most recent version of coupon $\zeta_{P,C}$ from its local coupon storage. If this is the first transaction between both interaction partners, it issues a new coupon. Subsequently, it updates the corresponding fields of $\zeta_{P,C}$, according to the parameters that are specific to this transaction. That is, it fills the coupon's first three data fields with the identities of P, C, and M, respectively, increments the coupon's version number, and adds the object size of O to the total amount of data transferred from P to C. After the coupon is updated, it signs $\zeta_{P,C}$ and symmetrically encrypts O using a secret session key K, in order to obtain the ciphertext $O_K = e_K(O)$. Furthermore, it computes $K_M = E_M(L, K)$ and creates the signature $Sig_P(L, K_M)$. Finally, the provider sends message flow (e2) to the consumer (cf. Table 9).

Upon receiving (e2), the consumer first verifies whether all entries of $\zeta_{P,C}$ are correct, including the identity of the mediator and the updated data volume. Also, it checks whether the provider's signature $Sig_P(\zeta_{P,C})$ can be verified correctly. Subsequently, it computes $h(O_K)$ and $h(\zeta_{P,C})$ to determine the unique transaction label L. This allows the consumer to verify whether $Sig_P(L, K_M)$ is the provider's signature on (L, K_M). Specifically, if the signature is valid, the consumer is convinced that either P or M can reveal the secret K to it. If any test fails or the consumer simply does not want to respond, it may quit the transaction without any liability. Otherwise, the consumer signs the coupon as well as the provider's signature and sends this signature back to the provider in step (e3).

As soon as the provider obtains flow (e3), it verifies whether $Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$ is indeed valid. If this is true, P has got the appropriate receipt for its service provisioning and therefore reveals the secret key K in message flow $(e4)$. However, if provider P does not receive message flow (e3) in time (according to P's definition), or consumer C

9 In Section 5.7.2, we present different ways on how such a mediator can be determined for the BitTorrent system.

(e1)	$C \longrightarrow P$	:	$h(O)$
	P	:	`if` interested in service provisioning `then` issue updated $\zeta_{P,C}$
(e2)	$P \longrightarrow C$	:	O_K, $\zeta_{P,C}$, $h(K)$, K_M, $Sig_P(\zeta_{P,C})$, $Sig_P(L, K_M)$
	C	:	`if` signature checks fail or $\zeta_{P,C}$ is incorrect, `then` stop
(e3)	$C \longrightarrow P$	:	$Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$
	P	:	`if` $Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$ is invalid `or` flow (e3) does not arrive in time, `then` run the abort protocol
(e4)	$P \longrightarrow C$	:	K
	C	:	`if` K is invalid `or` flow (e4) does not arrive in time, `then` run the recovery protocol
	C	:	`if` data encryption fails, `then` run the complaint protocol

Table 9: The Exchange Protocol.

has provided an incorrect signature, the provider may run the abort protocol to cancel the transaction.

When flow (e4) arrives, the consumer first computes $HK = h(K)$ and checks whether $HK \equiv h(K)$ as received in step (e2). If this equality holds, the consumer has ultimately obtained the secret key K to decrypt O_K. However, if C does not receive key K timely or K is invalid (i.e, $HK \neq h(K)$), it may run the recovery protocol to reconstruct key K with the aid of the mediator. Once C is in possession of key K, it computes $O' = d_K(O_K)$ verifies $h(O) \equiv h(O')$. If this is true, the protocol finishes. If $h(O) \neq h(O')$, however, the consumer may request help from the mediator by initiating the complaint protocol.

5.5.5 *Abort Protocol*

The abort protocol enables the provider to cancel a transaction that is indexed by label L, as soon as it does not receive the evidence of receipt (the consumer's signature) in time. The reasons for this can be different; *(i)* message flow (e3) is lost due to a communication failure; *(ii)* the consumer does not want to reply; or *(iii)* the message arrives, but the consumer's signature cannot be successfully verified. In either case, to protect the provider from getting exploited from a potentially misbehaving consumer, it must perform the following steps.

First, the provider issues the abort receipt $AR = Sig_P(\text{'abort request'}, L)$ that indicates that transaction L needs to be aborted. As shown in Table 10, this receipt is then sent to the mediator in message flow (a1), together with the other information that is necessary to cancel L.

Upon receiving flow (a1), the mediator first verifies whether the provider's signature on the abort receipt is valid. If this is not the fact, the abort request is simply ignored. If it is valid, M computes the unique label L to check its local database whether this protocol instance has previously been recovered by the consumer. If this is true, the mediator must be in possession of the consumer's signature which is immediately sent to provider P.

However, if L is not recorded, the mediator signs the abort receipt of the provider, issues the coupon revocation receipt $REV_{\zeta_{P,C}}$, and records (L, $state = aborted$) together with the information of flow (a1) in its local database. Subsequently, it confirms P that the transaction has been successfully aborted by sending $Sig_M(AR)$ in message flow (a2). At the same time, $REV_{\zeta_{P,C}}$ is sent to C in message flow (a3). This receipt can then be submitted to third-parties to revoke the latest version of coupon $\zeta_{P,C}$.

(a1)	$P \longrightarrow M$	:	$AR = Sig_P(\text{'abort request'}, L)$, P, C, $h(O)$, $h(O_K)$, $h(K)$, $\zeta_{P,C}$
	M	:	`if` AR is invalid, then stop
	M	:	`if` $(state = recovered)$, `then` fetch $Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$ from local database
			$M \longrightarrow P : Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$ and stop
	M	:	`else` set $(state = aborted)$ and issue $REV_{\zeta_{P,C}}$
(a2)	$M \longrightarrow P$	:	$Sig_M(AR)$
(a3)	$M \longrightarrow C$	:	$REV_{\zeta_{P,C}}$

Table 10: The Abort Protocol.

(r1)	$C \longrightarrow M$	:	P, C, $h(O)$, $h(O_K)$, $h(K)$, $\zeta_{P,C}$, K_M, $Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$, $Sig_P(L, K_M)$
	M	:	`if` signature checks fail, `then` stop
	M	:	`if` $(state = aborted)$, `then`
			$M \longrightarrow C : REV_{\zeta_{P,C}}$ and stop
	M	:	`if` K_M is invalid, `then`
			$M \longrightarrow C : REV_{\zeta_{P,C}}, FA_P$ and stop
	M	:	`else` set $(state = recovered)$
(r2)	$M \longrightarrow C$	:	K
(r3)	$M \longrightarrow P$	:	$Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$

Table 11: The Recovery Protocol.

REMARK: Note that the sole intention of message flow (a2) is to convince provider P that the protocol instance L has been successfully aborted. From there on, it is no longer possible for consumer C to run the recovery protocol, i.e., in order to obtain the secret key K. On the other hand, the sending of $REV_{\zeta_{P,C}}$ to C in flow (a3) is mandatory to counteract the following two scenarios, where provider P may illegally obtain a valid coupon: (1) After honest P has successfully aborted transaction L, it still receives flow (e3) due to a communication delay; or (2) upon receiving the valid coupon in flow (e3), a dishonest P immediately executes the abort protocol. Through this threat, P would obtain the receipt for its service provisioning whereas C has no opportunity to decrypt O_K.

5.5.6 *Recovery Protocol*

The main functionality of the recovery protocol is to reconstruct the provider's secret key K with the help of the mediator, once consumer C does not receive message flow (e4) timely or correctly. This situation is particularly annoying for C, because the service provisioning of provider P is already attested with message flow (e3). That is, P possesses C's signature on the latest version of $\zeta_{P,C}$, but C is unable to decrypt O_K. To still prevent that consumer C goes empty-handed, the recovery protocol is specified as follows (cf. Table 11).

After receiving message flow (r1), the mediator first computes label L, and verifies whether $Sig_P(L, K_M)$ and $Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$ are correct signatures. If both signatures are valid, the mediator inspects its local database to see if L has previously

(c1)	$C \longrightarrow M$	:	$P, C, h(O), h(O_K), h(K), \zeta_{P,C}, O_K, K_M, Sig_P(L, K_M)$
	M	:	`if` signature checks fail, `then` stop
	M	:	`if` $h(O') \neq h(O)$ `then`
			$M \longrightarrow C : REV_{\zeta_{P,C}}, FA_P$
	M	:	`else` do nothing

Table 12: The Complaint Protocol.

been aborted. If this is true, M retrieves $REV_{\zeta_{P,C}}$ from its local database and sends it to consumer C.

However, if the protocol instance has not been aborted before, the mediator inspects the correctness of K_M. To this end, K_M is correctly prepared if and only if $(K', L') = D_M(K_M)$, $h(K') = h(K)$, and $L' = L$. If these equalities do not hold, it is evident that peer C is victim of a fraud attempt of P. Accordingly, the mediator issues $REV_{\zeta_{P,C}}$ and FA_P and sends both receipts to the consumer. As mentioned previously, receipt $REV_{\zeta_{P,C}}$ allows node C to revoke $\zeta_{P,C}$, while receipt FA_P can be submitted to third-parties, proving that dishonest provider P has cheated.

On the other hand, if K_M is indeed valid, the mediator records $(L, state = recovered)$ together with the information of flow (r1) in its local database. After that, it reveals the secret key K to the consumer through message flow (r2). In addition to this, the provider obtains the consumer's evidence of receipt in message flow (r3).

Remark: Once the recovery request is successfully completed, it is intended that M forwards C's signature on $\zeta_{P,C}$ to P, thereby attesting P's service delivery. Otherwise, upon receiving flow (e2), a dishonest node C could always get the secret key K by immediately running the recovery protocol, without sending the corresponding evidence of receipt. Finally, in both the abort and the recovery protocol, we attach importance to the fact that the mediator is only provided with the hashed value of O_K, instead of receiving the entire ciphertext. This increases the privacy and efficiency of the approach, because *(i)* without the ciphertext O_K the mediator cannot derive O, and *(ii)* transferring a 160-bit long hash value consumes significantly less communication overhead than sending the entire ciphertext.

5.5.7 *Complaint Protocol*

Consumer C can execute the complaint protocol, if it receives the secret key K, either directly through provider P in message flow (e4) or with the aid of the mediator in message flow (r2), but it turns out that $h(e_K(O_K)) \neq h(O)$. To solve this conflict, the consumer has to proceed as depicted in Table 12.

After receiving message flow (c1), the mediator computes label L to test whether the provider's signatures on (L, K_M) can be correctly verified. After that, the mediator computes $(K, L) = D_M(K_M)$ to decrypt O_K: $O' = d_K(O_K)$. Finally, it tests whether $h(O') = h(O)$. If it turns out that this equality does not hold, it is proven that the provider has cheated the consumer. In this situation, the mediator issues receipt $REV_{\zeta_{P,C}}$ and FA_P, which are both sent to C. Otherwise, the provider has delivered the correct data as well as the proper secret key. In this case, the mediator simply ignores the consumer's complaint.

5.5.8 *Summary*

In the following, we briefly summarise how our devised data exchange specification meets the requirements posed in the beginning of this section.

First of all, if both parties follow the specification of the exchange protocol, the provider receives a valid coupon $\zeta_{P,C}$ for its data provisioning. While, at the same time, the consumer obtains the desired data object O. Importantly, this is achieved without the involvement of a mediator. Therefore, the devised solution is optimistic by relying on an *on-demand* mediator. In addition to this, our specification satisfies the following security properties. For conciseness and simplicity of presentation, however, we omit proofs on why these properties hold. The interested reader can found them in Appendix A.3.

Lemma A.3.1 For a dishonest consumer C, it is impossible to obtain data object O, without sending the correctly prepared evidence of receipt.

Lemma A.3.2 For a dishonest provider P, it is impossible to obtain a valid coupon $\zeta_{P,C}$, without sending the correct encryption key K.

Lemma A.3.3 For a dishonest provider P, it is impossible to upload garbage data, and yet obtain a valid coupon $\zeta_{P,C}$ for this illegal service provisioning.

To put it another way, Lemma A.3.1 guarantees that a service provider always receives a valid coupon for its proper service provisioning. While Lemma A.3.2 and A.3.3, on the other hand, say that a consumer always receives the data it has requested, once it has attested the receipt of the data. Therefore, the devised data exchange specification ensures that *exchange fairness* and *data integrity* is always guaranteed, even if one party tries to cheat the other. Note that in the special case in which both parties are dishonest, the protocol implicitly ends fairly; in this particular situation, it is assumed that none of the parties prepare the correct items, and, consequently, neither party gets an advantage over the other.

Finally, *timeliness* is guaranteed because the provider as well as the consumer can terminate the protocol at any time, while the other party can always run one of the three sub-protocols (i.e., the abort, recovery, and complaint protocol) to still achieve exchange fairness.

5.6 SERVICE POLICIES

A common property of currently deployed P2P content distribution systems is that peers are only connected to a small subset of neighbours. Out of these, a peer has to choose an outdegree k — the number of neighbours to serve simultaneously with data. In FairSwarm.KOM, the selection of those peers is regulated by a so-called *service policy*.

Depending on the instantiation of such a policy, different objectives can be pursued. For instance, system designers can establish an incentive compatible fairness norm, i.e., peers receive only as much as they give and high contributors are more likely to be chosen for uploading to. In contrast to this, if fairness properties are secondary or even not important at all, a service policy can also be tailored to optimise overall system throughput.

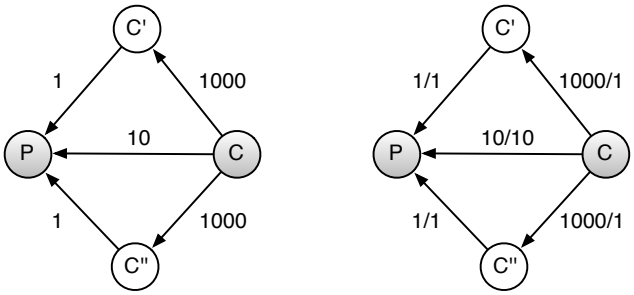
Thus, through the decoupling of the mechanism that prescribes *how* to use reputation (the Service Policy) from the mechanism *providing* this information (the Coupon Propagation Specification), system designers can freely evolve their own service policy depending on the environment the system is operating in. In the following, we first propose FairSwarm.KOM's *default* service policy, designed:

- to incite users to maintain their identities because the long-term persistence of user accounts is beneficial from a rational point of view.
- to meet the fairness-performance dilemma that arises due to peer heterogeneity. Specifically, to provide service differentiation by rewarding peers proportional to their upload contributions.
- to provide increased robustness against untruthful peer behaviour such as collusion and Sybil attacks.

Since system designers or individual users do not have to follow this recommendation, we also present a few variations including a brief discussion on their advantages and potential drawbacks.

5.6.1 *Default Policy*

While our data exchange specification protects a peer from getting cheated by a dishonest node, it cannot prevent a group of colluders creating and distributing coupons to each other, without having exchanged any data at all. Moreover, since our system design does not assume any centralised authority infrastructure to which users must register, this attack can also be performed by a single peer, using multiple zero-cost identities. In literature, this threat is often referred to as *Sybil attack* [26].



(a) *Without MaxFlow* (b) *With MaxFlow*

Figure 41: The effectiveness of the Sybil attack without and with the MaxFlow algorithm as countermeasure. Note that C′ and C″ are Sybil/collusion nodes under the control of service consumer C. These nodes artificially increase C's account balance, i.e., they generate and distribute coupons, without having exchanged any data at all.

If unaddressed, collusion or even Sybil attacks can heavily impair the correctness of an indirect reciprocation scheme. In particular, FairSwarm.KOM uses direct experiences as well one-hop contributions via intermediate nodes to evaluate the cooperativeness of a requesting service consumer C. Therefore, without any countermeasure, any service provider P would simply accept the coupons of consumer C, generated with common intermediate nodes, e.g., intermediate peer C′ and C″ in Figure 41a. However, if a dishonest C is in control of these two nodes (e.g., due to peer collusion or a Sybil attack), node C can attest itself arbitrarily high contribution values. It is apparent that this attack can significantly boost node C's standing in the system, making it likely to succeed against all the other peers simultaneously competing for downloading slots.

While Cheng et al. [19] has shown that incentive mechanisms relying on third-party information (i.e., indirect reciprocation schemes) can never defend completely against these kind of attacks, FairSwarm.KOM utilises the max-flow min-cut theorem to reduce their impact [37]. In particular, if a service provider P wants to compute the total upload contribution of a candidate peer C, it cumulates the amount of services that flow directly and indirectly (via one hop) from C to itself. To see this, consider

Algorithm 1 The default service policy of peer P.

Require: requesting_peers[], downloading_peers[]

1: **for all** consumer C requesting service **do**
2: **if** C $\notin$ downloading_peers[] **then**
3: $AccountBalance_C \leftarrow$ Computed using Equation 5.1
4: candidates $\leftarrow$ (C, $AccountBalance_C$)
5: **else**
6: Ignore request
7: **end if**
8: **end for**
9: C_{max} = max(candidates)
10: downloading_peers[] $\leftarrow C_{max}$
11: **return** C_{max}

Figure 41b. In this graph, the directed edges are labelled with two values. The first value is obtained from the data field of the corresponding coupon, prescribing the peer's long-term contribution. The second weight, on the other hand, is the fraction of the maximum flow from C to P, when applying the max-flow min-cut theorem.

From the figure, it is noticeable that irrespective of how many Sybil nodes peer C controls, it can claim only those contributions from provider P that each Sybil node has *directly* provided to P. This mitigates the issuing of coupons attesting fake contributions, as exemplified by edge (C', C) and (C'', C). Since this approach can also be applied to compute a candidate C's total amount of downloaded data, provider P obtains a consumer's *account balance* by using the following formula,

$$AccountBalance(C)_P = flow(C \rightarrow P) - flow(P \rightarrow C). \tag{5.1}$$

After detailing this important countermeasure to restrict collusion and Sybil attacks, Algorithm 1 ultimately shows how FairSwarm.KOM handles competing requests for downloading slots. Our default policy enforces that a requesting peer C can obtain at most one out of k downloading slots of provider P (line 2). This prevents a user with a high account surplus from claiming all download slots of peer P for itself alone, thereby removing the possibility of other peers participating in the download process. In addition to this, it can be seen that our policy ranks peers according to Equation 5.1, while the peer with the maximum account balance is chosen for uploading (line 3 + 9). This gives users incentive to maintain their identities in the long term, as *(i)* one-hop contributions across time and torrents are inherently max-flow conformant and *(ii)* these interaction relationships can bootstrap cooperation between new peer pairs. In fact, the more common interaction partners two peers share (currently or in the past), the more contribution flows between these nodes exist. This potentially increases a peer's account balance and thus increases its chances of reciprocation. In contrast to this, when newly arriving in the system, a peer has no contribution history and must therefore wait until it receives the first chunks from other nodes to prove its cooperativeness.

5.6.2 *Alternative Policies*

Our default service policy is only one way to evaluate the cooperativeness of peers, based on the coupons gossiped from the nodes. However, there are many other alternative ways to realise such a policy, which are briefly introduced in the following.

When devising a service policy, a system designer has always to weigh-off the different challenges it seeks to address. These challenges include the delicate trade-off between performance and fairness, and the robustness against attacks such as free-

riding, whitewashing, collusion, and Sybil attacks. To date, no policy exists (including ours) that is capable of effectively addressing all these challenges at the same time. Instead, optimising for one property often comes at the expense of others. For instance, to bootstrap cooperation and find better peering partners (performance issue), the designers of BitTorrent accept the risk that free-riders could exploit the optimistic unchoke functionality. In fact, being susceptible to one attack is indeed a negative property. However, as exemplified by BitTorrent, the free-riding of some users is neither the cause for its current performance issues nor does it directly induce the system's break down [132].

Next, we detail three major classes of alternative service policies, each of them striving to achieve a certain goal.

Design Alternative 1: Optimising for Fairness

A policy that optimises for contribution fairness should keep track of the peers' share ratio in the system (the total amount of data provided divided by the total amount of data consumed). To this end, a peer is only considered as a potential candidate, if its share ratio is equal or above one. Intuitively, the approach would then select peers exhibiting the highest ratios for downloading. This enables that peers receive exactly as much as they give (which seems fair).

Introducing such a contribution threshold clearly incites users to continuously provide data to avoid being unable to download content at a later occasion. Further, guaranteeing contribution fairness can be an important incentive for users to participate in the system, especially in settings where Internet Service Providers charge users on uplink usage or uplink bandwidth is scarce. In those systems, asymmetries in terms of data provided and consumed should not be systematic.

The drawback of contribution thresholds is, however, that the peers' upload slots may remain unused once no candidate peer can be found exceeding these thresholds. Accordingly, the peers' upload utilisation may be far from optimum, which demonstrably hurts system throughput [31]. Furthermore, while a share ratio threshold complicates free-riding, peers have a clear incentive to 'whitewash' identities. In particular, a whitewasher could initially obtain a chunk, immediately leave the system and rejoin it with a new identity to repeat this procedure. Apparently, this attack is particularly effective in systems where identities are for free and peers do not need to register to a centralised authority.

Design Alternative 2: Optimising for Performance

A policy that optimises performance must pursue the goal that users (especially high capacity peers) are kept engaged as long as possible in the data distribution process [31, 11]. To achieve this in a distributed way, uploader selection must be non-discriminative [31]. That is, users must randomly select downloading candidates irrespective of a peer's account balance.

Consequently, when optimising for total system throughput, the demands of a particular user must be subordinated to the welfare of the system. This clearly suggests, however, that fairness cannot be guaranteed. For instance, that a peer that contributes more finishes earlier or that peers receive exactly what they give. Furthermore, non-selective uploading makes free-riding easily possible, because peers ignore the contribution histories of other nodes. Also, due to the same reason, the approach lacks incentives for seeders. Consequently, this policy is not suitable for P2P content distribution systems that attempt to achieve long-term availability of content. However, it is still appropriate for scenarios where a content provider wants to quickly replicate a single file object to a large number of users, e.g., a critical software update.

Design Alternative 3: Maximum Robustness against Attacks

As mentioned above, it is impossible to make a system based on third-party experiences resistant to all forms of strategic or malicious attacks [19]. The best one can do is to reduce the effectiveness of these attacks, and thus to apply a policy that is as conservative as possible.

To mitigate collusion or Sybil attacks, users must only consider coupons that were issued by themselves, i.e., due to direct interactions with a given peer. This means that information about third-party experiences will not be considered at all. Further, to hamper free-riding, a one-chunk deficit threshold should be introduced. That is, a peer can only download a file chunk if it provides one in return. This is particular effective in the current BitTorrent system, because a single file object is typically composed of thousands of data chunks. However, as shown by our large-scale measurements, a typical BitTorrent swarm amounts to less than 170 online peers. Thus, regardless of whether a free-rider obtains a chunk in the first interaction, it would have to wait a long time to eventually download the entire file.

Clearly, there is an trade-off between the first data provisioning to bootstrap cooperation and the long-term protection of participants against whitewashing [38]. To tackle this issue, peers should adopt an adaptive policy that treats newcomers based on the experiences they had with other newcomers in the past [58]. For instance, a peer would only initially cooperate if newcomers have continuously reciprocated in the past. This allows generosity when others are being generous, reasonably addressing the above trade-off.

While this service policy ensures strict fairness (i.e., the one-chunk deficit threshold is an extreme form of the policy optimising for contribution fairness) and is highly robust to attacks, care must be taken with regard to download performance. In particular, the users' upstream capacity remains unused, unless at least one interaction partner recoups its contribution deficit. This occurrence, however, can easily double download times, compared to the currently employed incentive strategies of BitTorrent [54]. Finally, as seeders would limit their contributions to at most one chunk, the approach is not suitable for environments where long-term availability of content is desired.

5.6.3 *Discussion*

Our default service policy represents an interesting trade-off between the different design extremes with regard to performance, fairness, and robustness.

First, our policy attempts to address the performance-fairness dilemma by preserving each of both properties to a certain extent. In particular, as seen in Algorithm 1, the default policy refrains from using contribution thresholds that a peer must exceed to obtain data. This ensures that the users' uplinks is always utilised, which is clearly in favour of system throughput. However, to still guarantee a reasonable level of fairness, peers do not perform non-discriminative uploading. Instead, the upload strategy is *selective*, meaning that users that have contributed more are likely to receive more downloading slots in the system (line 9). This clearly enables service differentiation proportional to the users' upload contributions.

To evaluate peers in the system, our default policy uses direct experiences as well as one-hop information. For that reason, it is vulnerable to collusion and Sybil attacks. However, as shown in our solution space analysis, using information beyond direct observations is a necessity to provide robust incentives for seeders (cf. Requirement 1 in Section 4.4). Thus, the best one can do is to reduce the impact of those attacks, which we address by applying the min-cut max-flow theorem.

In fact, the lack of contribution thresholds in FairSwarm.KOM's default policy makes the approach susceptible to free-riding. However, through the use of a selective uploading strategy, free-riding is highly complicated. Specifically, to obtain data, a free-

rider must win the auction for a downloading slot. However, since these kinds of users are typically characterised by an account deficit, this situation will occur rather seldom in practise. Furthermore, compared to free-riders, a whitewasher seems to be better positioned, since its account balance can be 'whitewashed' to zero. However, when doing so, it loses the opportunity to boost its account balance through one-hop indirect reciprocation, which hampers the competition against obedient nodes maintaining their identities in the long-term. We come back to these issues in Chapter 6, when we thoroughly study the robustness properties of our default policy against most common attacks of P2P systems.

5.7 IMPLEMENTING FAIRSWARM.KOM IN BITTORRENT

FairSwarm.KOM's data exchange specification as well as its unstructured way of propagating coupons in the system are orthogonal to the basic principles of P2P content distribution. Therefore, the approach can be easily deployed on top of any currently existing P2P content distribution system, without altering the system's basic functionality. In particular, in P2P content distribution, a transaction typically relates to the transmission of the entire file or even file chunks when using multi-source downloading. Accordingly, it is straightforward to extend a system's data exchange process with the specification provided in Section 5.5, in order to automate the generation of coupons for each data transfer. Second, in P2P content distribution systems, peers typically maintain overlay connections to a small subset of nodes (e.g, to exchange control traffic or to transfer user data). This communication infrastructure can be exploited by FairSwarm.KOM's coupon propagation algorithms to establish an indirect reciprocation infrastructure. Therefore, there is no need for an additional overlay network on top of the application, introducing additional complexity and overhead.

Consequently, when implementing FairSwarm.KOM in the BitTorrent system, only slight changes are necessary. More precisely, a basic design principle of FairSwarm.KOM is to enable indirect reciprocation without the need of any centralised authority infrastructure that mints identities, provides accounting, and punishes dishonesty. This goal aligns well with the usage model of BitTorrent that is also characterised by openness and no centralised trust. As a consequence, the usage of FairSwarm.KOM with BitTorrent does not require the introduction of new actors or other infrastructural components. Instead, the only thing that needs to be clarified is *(i)* how to enable long-term identification of users in the BitTorrent system, *(ii)* how to efficiently incorporate FairSwarm.KOM's data exchange and coupon propagation specification, and, lastly, *(iii)* how to provide the flexibility of supporting different service policies. In the following, we give answers to each of these issues.

5.7.1 *Identity Management*

In BitTorrent, peers do not have permanent identifiers (IDs). Instead, whenever a user starts a BitTorrent client, a 20 byte long random session identifier is generated. To still enable the long-term identification of peers in BitTorrent (which is a prerequisite for FairSwarm.KOM to work), the ID generation process should be changed as follows. First, to participate in the system, a user must initially generate a public/private key pair which is persistently stored on its hard disk. When connecting to the BitTorrent network, the user hashes its public key K_{pub} using SHA-1 (Secure Hash Algorithm 1) [125]. The output is a 160-bit string that represents the user's unique identifier in the system. Since SHA-1 belongs to the class of hash functions that is collision-free, it is impossible to obtain another public key K'_{pub} such that $h(K'_{pub}) = h(K_{pub})$. Consequently, once the public key is submitted to other users, i.e., for signature verification, these nodes can be sure that a message clearly originates from the node

Algorithm 2 Modified unchoking mechanism of BitTorrent.

Require: candidate_peers[], downloading_peers[], available_slots

```
1:  while available_slots > 0 do
2:    C_max ← apply Algorithm 1 to candidate_peers[]
3:    if C_max == null then
4:      break
5:    else
6:      available_slots ← available_slots −1
7:      downloading_peers ← downloading_peers.add(C_max)
8:      perform data exchange process with C_max (see Figure 42)
9:    end if
10: end while
```

belonging to a given ID. This simple principle enables not only the identification, but also the authentication of users in BitTorrent, without the need of any complex public key infrastructure.

5.7.2 *Unchoking*

In BitTorrent, unchoke decisions are performed in fixed time intervals. In particular, every 10 seconds, each user selects a predefined number of peers for sending content to. These peers are then allowed to download data until the next period of 10 seconds is up. In addition to this, both to probe for better peering partners and to bootstrap cooperation, every 30 seconds, at least one peer is randomly selected for optimistic unchoking. This time-driven model must be replaced with FairSwarm.KOM's transaction-based approach. In particular, FairSwarm.KOM requires that *(i)* upon the completion of every single transaction, both peers receive a coupon about the actual data exchange, and, after this, *(ii)* the occupancy of the respective downloading slot is re-evaluated according to the adopted service policy.

To meet these two requirements, Algorithm 2 presents a modified unchoking mechanism for BitTorrent. Before this algorithm is executed, a BitTorrent peer first determines the amount of available upload slots. This can simply be computed by subtracting the amount of downloading peers from a peer's outdegree k. Subsequently, for each available upload slot, the algorithm then determines a winner out of a set of candidate peers using a predefined service policy, e.g., FairSwarm.KOM's default service policy (line 2). Note that a peer belongs to the set of candidate peers if and only if its interested flag is set to true. To this end, we say a peer is interested in another one if it possesses at least one chunk that the interested peer wants to download. If no peer can be found, i.e., all the candidate peers are already occupying a downloading slot or none of them is interested, the algorithm stops (line 3 + 4). Otherwise, the data exchange process is initiated as illustrated in Figure 42 (line 8). To inform the chosen candidate peer about the possibility to download content, it receives an *unchoke* message from the data provider. The data consumer then replies with a *chunk* message containing the hashed index of the chunk it wants to download. From then on, everything proceeds according to the data exchange specification, described in Section 5.5.

An important issue for FairSwarm.KOM's data exchange specification is to find an appropriate mediator that is independent from a given peer pair, exchanging content. In the BitTorrent system, this can be achieved in different ways. For instance, one alternative is to use a peer whose identifier is 'closest' to the hashed valued H_C of the chunk to be transferred. Since a peer's identifier is calculated by applying a collision-free hash function to its corresponding public key, it is impossible to generate a public key K_{pub} such that $h(K_{pub}) = H_C$. Thus, none of both parties may influence the

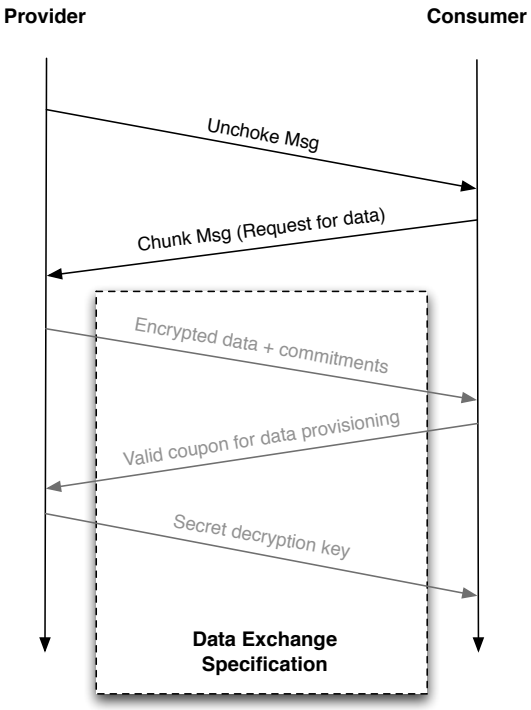


Figure 42: Data exchanges in BitTorrent when incorporating FairSwarm.KOM's data exchange specification.

choice of the mediator in its favour. On the other hand, since a mediator cannot infer detailed information about a peer's transaction history (i.e., when and which content a peer has downloaded exactly), we recommend to delegate this task to the entry point of the system, namely the *tracker* of a BitTorrent swarm. By doing so, a user can directly obtain the mediator's public key when connecting to the system. Moreover, a tracker can be considered as more reliable and long-lived, compared to ordinary peers.

5.7.3 *Propagation of Coupons*

So far it has been discussed how the process of issuing coupons can be incorporated into BitTorrent. To establish FairSwarm.KOM's one-hop reciprocation infrastructure, on the other hand, these coupons must be disseminated in the system, according to the specification presented in Section 5.4. This is, however, straightforward in BitTorrent. In particular, every peer typically maintains a number of TCP connections to a subset of the peers participating in a given BitTorrent swarm. These overlay neighbours are potential recipients and providers of data. This is inline with FairSwarm.KOM's coupon propagation specification that only distributes coupons to those nodes that are interaction candidates. Thus, there is no necessity to create any additional overlay connection for the dissemination of coupons. Instead, FairSwarm.KOM's coupon propagation specification can simply exploit the already existing communication infrastructure of BitTorrent.

5.8 SOLUTION REQUIREMENTS REVISITED

Section 4.4 has provided a set of key requirements that a more sophisticated incentive mechanism must meet to improve the performance and availability properties of the BitTorrent system. This section delineates how these requirements have been incorporated into our proposed solution approach.

- **Requirement 1** (Multilateral Incentive Design). *The solution must be designed for detaching incentives from time, content, and individual peers.*
  - FairSwarm.KOM is an indirect reciprocation scheme that allows a user to contribute data to a peer X at a given time and receive reciprocation from another peer Y at another point in time (via one-level of indirection), without encountering this node before. More importantly, this is regardless of what swarm these users operate in (cf. Section 5.4). As such, the approach is agnostic to time, content, and individual users. Consequently, Requirement 1 is clearly fulfilled by FairSwarm.KOM.

- **Requirement 2** (Decentralisation). *The solution must be designed to operate without the involvement of any pre-trusted or centralised authority infrastructure.*
  - FairSwarm.KOM uses coupons (bidirectional transaction receipts) that are propagated in the system to enrich the knowledge base of the nodes. This is the only source of information that peers use to evaluate the cooperativeness of others. As such, the approach is fully decentralised and does not require any pre-trusted or centralised authority infrastructure, thereby meeting Requirement 2.

- **Requirement 3** (Robustness to Impersonation). *The solution must be designed to hinder a malicious peer to adopt the identity of other participants in the system, thereby preventing misbehaviour in the name of the victim.*
  - Since coupons contain the unique identifiers of their producers, and are protected through their signatures, it is hardly possible for a third-party to misbehave in the name of another user in the system. This fulfils Requirement 3.

- **Requirement 4** (Robustness to Rational Attacks). *The solution must be designed to provide robust countermeasures against peer collusion, whitewashing and the Sybil attacks such that the benefit from running one of these attacks is limited.*
  - This solution requirement has been addressed through the proposal of FairSwarm.KOM's default service policy. In particular, users who continuously change their identities (i.e., whitewashers) lose an important opportunity to boost their account balance through one-hop indirect reciprocation. This hampers the competition against obedient nodes maintaining their identities in the long-term. Furthermore, the min-cut max-flow theorem is used to mitigate peer collusion and Sybil attacks.

- **Requirement 5** (Robustness to Churn). *The solution must be designed to allow peers to reliably store and lookup experiences about third-party behaviour, irrespective of whether some nodes suddenly crash or leave forever.*
  - Since FairSwarm.KOM limits the propagation of coupons to at most one hop in the overlay network, the sender of a coupon is always its producer. Thus, no information can be lost due to churn, because peers persistently store their own coupons in their local database. This clearly meets Requirement 5.

- **Requirement 6** (Self-sustainability). *The solution must be designed to incite users for propagating protocol relevant messages.*

  - In FairSwarm.KOM, peers have a clear incentive to distribute coupons to their neighbours. More precisely, through the use of one-hop indirect reciprocation, one of both coupon producers (i.e., the peer with an account surplus) is always interested in submitting a given coupon to third-parties. This improves its standing in the system and increases its chances to receive reciprocation. As such, the gossiping of coupons and, thus, the establishment of an indirect reciprocation infrastructure is inherently self-sustaining in FairSwarm.KOM, thereby fulfilling Requirement 6.

- **Requirement 7** (Contribution Fairness). *The solution must be designed to provide service differentiation that is proportional to the upload contribution of the participants.*
  - This solution requirement has been addressed through the proposal of FairSwarm.KOM's default service policy. The uploading strategy of this policy is selective, meaning that users that have contributed more are likely to receive more downloading slots in the system.

Intuitively, Requirement 4 and 7 cannot be formally verified and quantified, and will therefore by critically analysed in the evaluation of FairSwarm.KOM (see Chapter 6).

5.9 CONCLUSIONS

This chapter has presented the newly devised P2P incentive mechanism, named FairSwarm.KOM. First, a conceptual overview of FairSwarm.KOM's principle elements was given. Following this, each of these elements was presented in detail, while major design decisions were justified. Though the resulting P2P incentive mechanism is a generic solution that can be adapted to a variety of P2P content distribution systems, it was detailed how FairSwarm.KOM can be implemented in the BitTorrent system. Finally, the solution requirements of Section 5.4 were revisited to show how they have been taken into account by FairSwarm.KOM's design. The key features of the resulting mechanism can be summarised as follows:

- FairSwarm.KOM is fully decentralised and self-sustainable (i.e., peers always have an incentive to follow the protocol specification).
- In contrast to currently deployed incentive mechanisms (i.e., tit-for-tat), FairSwarm.KOM allows peers to evaluate others by using information that goes beyond local histories and direct observations. As such, it is a multilateral incentive scheme that is agnostic to time, content, and individual peers.
- FairSwarm.KOM exploits the long-term interaction pattern of the users and limits the propagation of third-party experiences to at most one-hop in the overlay network. This fosters scalability and provides increased robustness (i.e., to churn and attacks).
- FairSwarm.KOM employs a novel coupon-based approach to persistently document the bilateral interaction pattern of two peers over time. This enables one-hop indirect reciprocation across time *without* the need for requesting intermediate nodes to attest contributions (i.e., as performed in recent works such as [94, 15]). This is highly important as these intermediaries are offline in the majority of cases (71%).
- FairSwarm.KOM decouples the mechanism prescribing how to use reputation (the Service Policy) from the mechanism providing this information (the Coupon Propagation Specification). This allows to cope with the performance-fairness dilemma in P2P content distribution (cf. Section 1.3).

To conclude, through the development of FairSwarm.KOM, the second research goal of this thesis has been addressed (cf. Section 1.4). The next step to take is to evaluate FairSwarm.KOM's strengths and weaknesses when integrated into the BitTorrent system.

6 EVALUATION OF FAIRSWARM.KOM

Upon presenting the design of the devised incentive mechanism FairSwarm.KOM, this chapter now focuses on the evaluation of this scheme. Our goal is to validate how FairSwarm.KOM overcomes the limitations of currently deployed incentives strategies (i.e., tit-for-tat) and therefore offers improved performance and availability properties to the BitTorrent system.

6.1 INTRODUCTION

So far, the basic principles and algorithms of FairSwarm.KOM have been detailed. To this end, it has been also explained how FairSwarm.KOM's design incorporates the requirements outlined within Section 4.4. Based on these requirements, the next step is to evaluate the efficiency of the approach. In the following, we begin by stating our concrete evaluative aims. Since a number of different approaches could be taken, we then delineate our evaluation methodology. Subsequently, we present the results and insights obtained, and conclude with a critical evaluative summary.

6.1.1 *Evaluative Aims*

The overarching goal of this evaluation is to find the strengths and weaknesses of FairSwarm.KOM with a focus on performance, availability, fairness, overhead, and robustness. In particular, we wish:

- to measure and quantify the potential *performance* and *availability* benefits that FairSwarm.KOM can offer BitTorrent.
- to ascertain if FairSwarm.KOM can improve BitTorrent's performance and availability without damaging user and system-wide *fairness*.
- to measure the *overheads* related to utilising FairSwarm.KOM under current environments and workloads.
- to validate that FairSwarm.KOM is *robust* against common P2P attacks and to ensure that malicious users do not achieve superior performance to those that adhere to FairSwarm.KOM's policies.

6.1.2 *Evaluation Methodology*

An extensive analysis of P2P incentive mechanisms is often very complex, since many important aspects must be considered at the same time. For instance, it is insufficient to study only one system property (e.g., availability), but neglect other issues such as client download performance, fairness, overhead, and robustness to attacks. This is because each of these properties directly influences each other, and optimising one may come at the expense of another (cf. Section 5.6). In addition to this, quantifying the effectiveness of incentive strategies for promoting long-term incentives for seeding – a key requirement to overcome the availability and performance issues of current P2P content distribution systems – requires a workload model that must itself be long-term and lifelike. That is, it must capture the user behaviour over a prolonged period of time as well as other characteristics describing the operational environment of the system.

To reasonably meet these evaluation challenges, we utilise our rich long-term measurement data of the BitTorrent system and perform a threefold analysis of FairSwarm.KOM:

- *System Parameter Tuning.* First, we study important system parameters of FairSwarm.KOM for controlling overhead, fairness, performance, and robustness properties of the approach. Here, our goal is to obtain an appropriate setup of parameters for BitTorrent-like workloads, while finding a good balance between the above properties.

- *Comparing P2P Incentive Mechanisms.* Upon assigning values to important system parameters, we compare FairSwarm.KOM with regard to performance, availability, fairness, and overhead to the tit-for-tat incentives of BitTorrent. Additionally, since BitTorrent uses solely local observations for decision making, while FairSwarm.KOM additionally incorporates experiences of one-hop neighbours, we also compare these two incentive approaches to an artificially assumed shared history system that has *global* knowledge about all occurred interactions in the network. It is intuitive that such an oracle-like incentive approach can never exist in reality. However, we are interested in finding out how much performance, availability, and fairness could improve compared to (deployable) approaches that use less information for decision-making. Therefore this oracle-based shared history mechanism represents an upper bound for the information that can be available at the peers, and thus constitutes a second baseline for FairSwarm.KOM.

- *Robustness to Untruthful Peer Behaviour.* Lastly, we test the robustness of FairSwarm.KOM against dishonest users attempting to gain an advantage in the system. Specifically, since the usage of third-party experiences greatly expands the range of possible attacks to strategising and malicious users, this analysis comprises *(i)* attacks that are common in already existing P2P content distribution systems (i.e., free-riding) and *(ii)* more sophisticated attacks that may arise due to the sharing of information among the peers.

Note that FairSwarm.KOM is an indirect reciprocation protocol (incentive mechanism) that is intended to operate on top of an already existing P2P content distribution system that provides a content delivery and discovery component. Thus, when referring to the term *FairSwarm.KOM* in the remainder of this chapter we implicitly assume that FairSwarm.KOM operates on top of the basic functionality provided by BitTorrent. Accordingly, FairSwarm.KOM only replaces the incentive mechanism of BitTorrent by using its own mechanisms and protocols to foster cooperation (cf. Section 5.7).

6.2 SYSTEM PARAMETER TUNING

The coupon propagation specification of FairSwarm.KOM defines precisely when and whom to send coupons to, thereby establishing an indirect reciprocation infrastructure (cf. Section 5.4). However, we have deliberately delayed the assignment of two workload-dependent parameters: *(i)* the percentage threshold for coupon updates and *(ii)* the total amount of probabilistic coupons to send during bootstrapping. We now study these parameters to find an appropriate system setup for the comparative analysis following in the next section. Before presenting, however, the results obtained, we first detail our methodology.

6.2.1 *Experimental Methodology*

Each of the two system parameters under study requires a specific experimental methodology. Concretely, the first tuneable parameter, the percentage threshold for coupon updates, can only be studied when peers are truly exchanging data. This is because the update period of coupons solely depends on the frequency and intensity of data transactions between the peers. Since it is highly challenging to setup live experiments with several thousands of users exchanging only a single file, we opt for a

Context	Parameter	Value
Simulated environment	Number of Leechers	1,000
	Number of Seeders	10
	Client bandwidth distribution	Trace-based [50]
	File size (in MB)	10, 100, 750
	User Arrival Time	Flash-Crowd
FairSwarm.KOM parameters	Percentage threshold (in %)	0, 10, 20, 30, 40, 50
	Peer outdegree k	5
	Max. number of overlay neighbours	50
	Service Policy	default

Table 13: Overview of experiments with different percentage thresholds for coupon updates.

simulation-based approach to evaluate the influence of this parameter. To achieve this, we use the simulation tool delineated in Section 4.3.1 that has been extended to use the FairSwarm.KOM protocol.

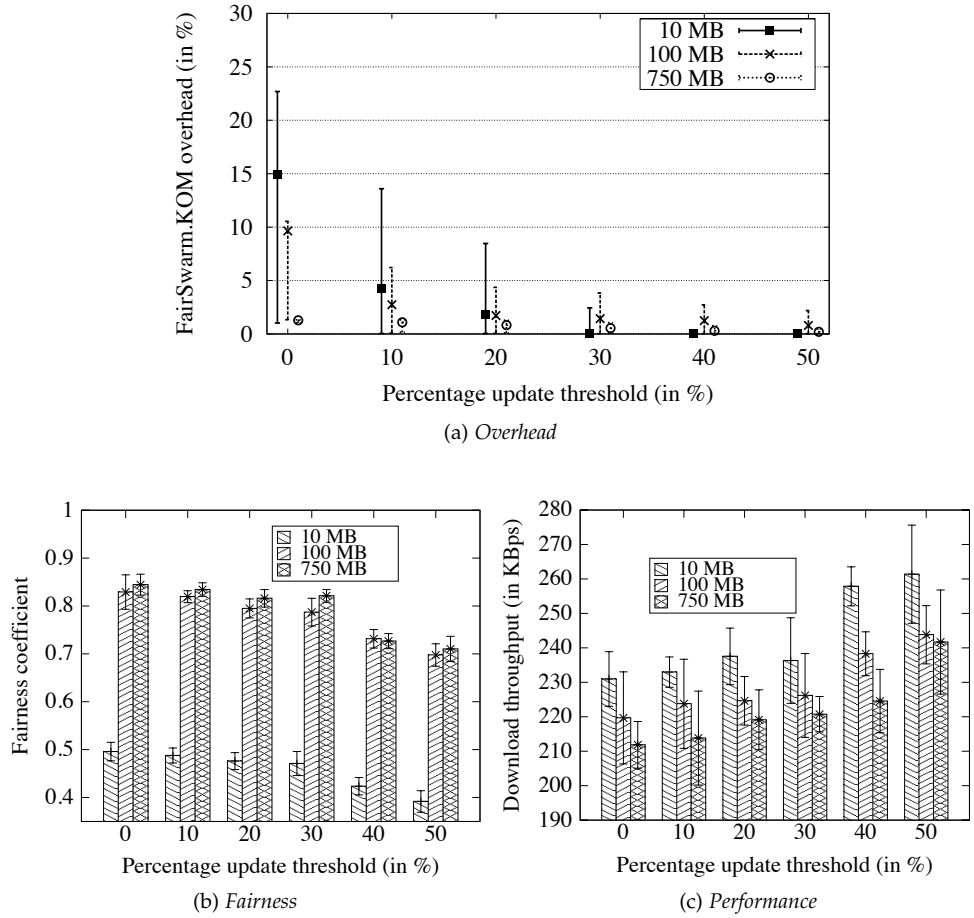
For the second system parameter, on the other hand, we seek to determine the total number of coupons to gossip when establishing a neighbourhood connection to a new peer. The concrete value of this parameter clearly depends on the users' prior interaction partners in the past. Therefore, to ascertain this we directly inspect our long-term measurement data of the BitTorrent system. This data allows us to ascertain which torrents a given peer has participated in and whom it could have interacted with.

6.2.2 *Percentage Threshold for Coupon Updates*

The first parameter that needs to be assigned is the percentage threshold for coupon updates. This parameter determines the frequency of coupon updates and, thus, controls the trade-off between information up-to-dateness and communication overhead. Especially the former can be important for decision making, as up-to-date information allows peers to recognise and reward contributions of other users in a fair way.

To see what impact different parameter values have on fairness and communication overhead, we consider a few variations in our simulation-based study. Table 13 presents an overview of the different experiments. For each experiment, we assume a flash-crowd scenario in which 1,000 leechers attempt to download a single file object. This file is served from 10 seeders such that enough aggregate system capacity is available to replicate the file with exponential capacity of service [71, 129][1]. In order to have a representative client bandwidth distribution for leechers and seeders, we assign the access links of the users according to the measurement data of Isdal et al. [50]. This data is taken from more than 100,000 BitTorrent users and is thus to be considered as very representative for the current BitTorrent system. Furthermore, the number of overlay neighbours to whom a peer connects to as well as the peer's outdegree k (the amount of users to whom a peer simultaneously uploads) is directly derived from the BitTorrent system [21]. This is because FairSwarm.KOM operates on top of BitTorrent's content delivery component, meaning that content-delivery specific settings that are optimised for BitTorrent can be also directly taken for FairSwarm.KOM. The only two parameters that have been varied in our experiments is the file size and the percentage threshold for coupon updates.

1 Studying such a flash-crowd scenario represents an extreme situation with regard to the system load that is sufficient for our needs to get a broader understanding on the impact of different parameter values.



(a) *Overhead*

(b) *Fairness*

(c) *Performance*

Figure 43: Influence of different percentage thresholds for coupon updates on overhead, fairness and performance. Error bars in Subfigure (a) are min-max values, while for Subfigures (b) and (c), we present 95% confidence intervals.

We study three evaluative metrics while investigating this parameter. The first one, the *percent overhead*, quantifies the communication overhead caused by FairSwarm.KOM. For each individual user, it is defined as follows: (aggregate amount of data sent and received due to FairSwarm.KOM / total amount of data sent and received at this node). The second one, the fairness *correlation coefficient* $\rho_{U,D}$, quantifies the relation between the upload rate of users (U) and their experienced download rate (D). This fairness metric has already been used in our solution space analysis in Section 4.3. Finally, since performance and fairness properties are closely tied together [31], we also measure the users' *download throughput*, defined by the total download amount divided by the download time.

In Figure 43, we present the overhead, fairness and performance results obtained from varying percentage threshold values and file sizes. All results except the fairness correlation coefficient are average values, computed over all participants. Further, for reasons of statistical representativeness, all experiments have been repeated 20 times with different random seed values.

We begin by inspecting the communication costs of FairSwarm.KOM, as depicted in Figure 43a. It is interesting to see that the overhead curve drops off exponentially when increasing the percentage threshold value. This can be clearly observed for small-sized and medium-sized content, while this trend flattens out for very large object sizes. The reason for the latter is that the control traffic produced by FairSwarm.KOM seems to

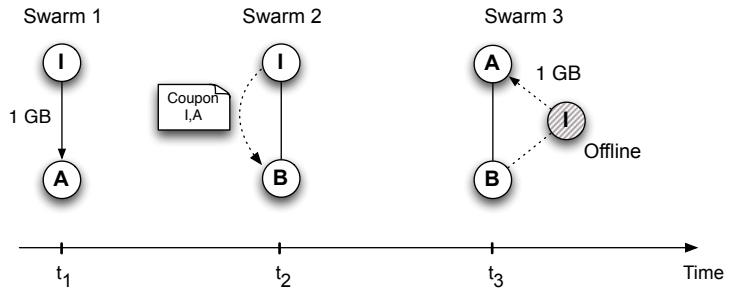


Figure 44: Gossiping of probabilistic coupons.

have little impact compared to the user data traffic generated by the content delivery component.

With regard to fairness, we observe for medium and large content sizes a strong correlation between the users' upload speed and download throughput, confirming superior fairness properties of the approach (cf. Figure 43b). We also observe positive fairness properties for smaller-sized content. Fairness is, however, reduced because download times are too low such that some peers have already finished their downloads, when others receive their first chunk [57]. Moreover, it can also be noted that increasing the value of the percentage threshold decreases the fairness for all file sizes. The direct consequence of this is that higher provisioned nodes must stay longer in the system to finish their downloads which positively influences the download times of the remaining users. As a consequence, decreasing levels of fairness results in improved average download throughput, as shown in Figure 43c.

Comparing the different results obtained, we set FairSwarm.KOM's percentage update threshold for the BitTorrent system to 30%. This is well-founded by the fact that the achieved overhead improvement for values higher than 30% is only marginal, while the chosen parameter value seems to represent a good trade-off between the two competing properties of fairness and performance.

6.2.3 *Number of Probabilistic Coupons*

Upon assigning a value to the percentage update threshold, we next focus on our second tuneable parameter: the number of probabilistic coupons. Specifically, when two peers encounter each other in FairSwarm.KOM, they mutually exchange *(i)* coupons about common intermediate nodes and *(ii)* coupons that are most favourable to either party (cf. Section 5.4). The gossiping of favourable coupons (or synonymously probabilistic coupons) is necessary to provide increased robustness against selfish users attempting to unfairly improve their standing in the system. Specifically, these dishonest users could suppress the existence of coupons that identify their previous high resource consumption (and potentially poor resource provision).

To mitigate this attack, the gossiping of probabilistic coupons is highly helpful as illustrated in Figure 44. In overlay swarm 1, dishonest peer A has obtained 1 GB of data from peer I. When peer B meets node I in overlay swarm 2 at a later date, assume that peer I proactively gossips coupon $\zeta_{I,A}$ to node B, because this coupon is one of the peer's most favourable ones. Subsequently, when B encounters dishonest peer A in overlay swarm 3, peer A may attempt to suppress the existence of intermediate node I, because the sending of coupon $\zeta_{I,A}$ would significantly decrease its standing at peer B[2]. That is, peer B would immediately know that I has previously consumed 1 GB of data in the system, and, thus, it is likely that peer B would prefer other nodes for the

2 Recall that this attack might also be worthwhile from a selfish user's point of view, as the probability that intermediate node I is online in the system is only 29% (cf. Section 5.4.2).

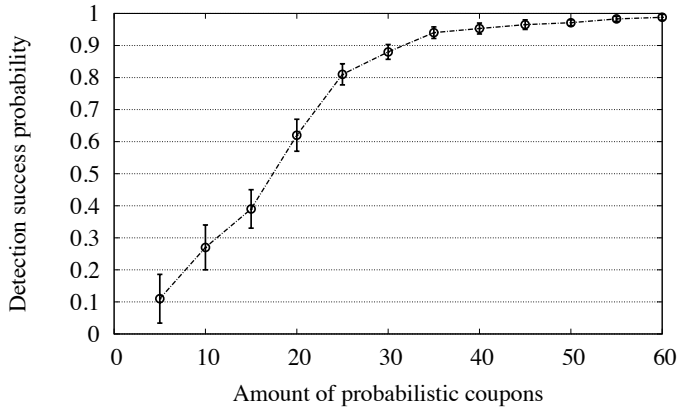


Figure 45: Success probability for the detection of fraudulent concealment of intermediate nodes. Error bars represent 95% confidence intervals.

uploading of content to. However, since peer B is already in possession of coupon $\zeta_{I,A}$, node A's fraud attempt will fail to succeed.

When considering this simple example, the fundamental question is *how many coupons must be proactively gossiped to successfully address fraud attempts such as of node A?* Ideally, to answer this question in an accurate way, one must know *when* exactly and *how much* data has been transferred between the different participants in the system. This is, however, impossible to infer without having access to the logs of all BitTorrent clients, residing at users' home PCs.

To remedy this issue, we utilise our long-term BitTorrent measurement data to determine the number of probabilistic coupons required. Specifically, through the use of the PEX protocol [103], we are able to infer that, on average, each user connects to 37 other participants per hour. Moreover, by using the technique described in Appendix A.1, we are able to determine the session times of users. These two aspects can then be combined to identify the interaction partners of the nodes, i.e., we periodically draw a random subset from the pool of online nodes of a given torrent. Furthermore, when two peers encounter each other (i.e., they become interaction partners), we assume that each party gossips a tuneable number of probabilistic coupons from previous or current interaction partners. These coupons are drawn randomly from the peers' local databases. Through these assumptions, we are able to compute the likelihood for a given peer pair (A, B) that peer B possesses at least one coupon that peer A may attempt to conceal.

Due to the computational and memory complexity of such an analysis, we cannot test the above for every peer pair in our trace data. Instead, we opt for drawing a representative number of peer pairs (89,239) from this data (each of them is randomly chosen). In Figure 45, we summarise the results obtained. The x-axis presents the total amount of probabilistic coupons that peers have gossiped in our experiments, while the y-axis shows the average success probability of detecting the aforementioned fraud attempt. In particular, we say that a detection was successful when peer B meets a given peer A, and B possesses at least one coupon $\zeta_{I,A}$ that is detrimental for peer A.

From the figure, it can be observed that the gossiping of 40 probabilistic coupons is enough to mitigate the concealment of common intermediate nodes with high probability (>95%). We therefore set this tuneable protocol parameter to 40 for BitTorrent-like workloads. Nevertheless, this number can easily be adapted at run-time (further increased if necessary), as the overhead for gossiping 40 coupons is still relatively low, compared to the sizes of file objects being transferred. For instance, transferring 40 coupons including the mandatory public keys of the corresponding nodes requires only $40 \cdot (168 \text{ bytes} + 128 \text{ bytes}) \,/\, 1{,}024 \approx 11.6$ KB.

6.3 COMPARING P2P INCENTIVE MECHANISMS

The previous section has provided values for the workload-dependent system parameters of FairSwarm.KOM. This section compares FairSwarm.KOM to BitTorrent's tit-for-tat, to see whether it can improve the performance and availability properties of the BitTorrent system. To this end, we have the goal that these properties must be achieved in a low overhead manner and without hurting the individual users' fairness.

As a second baseline to compare FairSwarm.KOM against, we make use of an oracle-based incentive mechanism that is identical to FairSwarm.KOM, but possesses global knowledge of the P2P network. In particular, FairSwarm.KOM uses direct observations, as well as one-hop experiences to evaluate the cooperativeness of the participants. In this regard our intention is to see how much performance, availability, and fairness could improve when using a *shared history* in which every interaction becomes globally visible. We notice, however, that we refrain from considering overhead aspects, as well as design issues to implement such an oracle approach in a reliable and robust manner in practise. This is out of scope of this thesis.

6.3.1 *Experimental Methodology*

When evaluating the different incentive mechanisms in live experiments (i.e., field study), it would be necessary to build-up user communities consisting of several thousands of participants that are solely running the corresponding protocol under study. Further, to see the long-term effects of the different incentive strategies with regard to file availability and performance, these users would need to be monitored over a prolonged period of time. Most importantly, however, in order to enable a fair comparison between the different incentive approaches, the characteristics of the overlay swarms considered (e.g., peer arrival times, user Internet access connection, file types, total amount of online users, etc.) must be identical for all experiments carried out. It is intuitive to see that the installation of such an evaluative environment is highly challenging in practise and goes beyond the scope of this thesis.

Instead, we opt for trace-based simulations using the OctoSim simulator, developed by Microsoft Research [9]. In particular we proceed in a similar manner to our solution space analysis (cf. Section 4.3) and take our long-term BitTorrent measurement data as input. This rich data comprises all the necessary information to obtain a long-term workload model that reflects a BitTorrent community in a lifelike manner. This model is then repeatedly used to assess the impact of the different incentive strategies in practise.

In the following we first present details about how this workload model is composed from this data. Subsequently, we describe how the different incentive strategies are modelled in the simulator and provide the parameter setup that applies to our experiments.

Workload Model

Here, we describe which aspects have been taken into account to obtain a workload reflecting a BitTorrent community in the long-term.

- *Selecting the torrents.* The macroscopic data set encompasses data from more than 46,000 overlay swarms, measured over a period of several weeks. This is far more than the simulator is able to handle. Hence, we chose a random subset of 100 torrents from the set of torrents affected by seedless states, with varying file sizes between 3 and 1,500 MB. This procedure has been repeated five times, resulting in five different torrent subsets with which we have experimented with. The logs of each of these subsets contain data of more than 45,000 downloads, distributed over at least four weeks.

- *User behaviour.* Similar to our solution space analysis, we do not use any artificial peer arrival function to model the content access pattern of the users. Instead, we bring up peers according to the trace logs. To model the number of swarms that a peer joins, we calculate the probability distribution over our entire data set (4.98 on average). When re-joining the network, FairSwarm.KOM users always persist as file replicas in torrents that they have previously downloaded. Furthermore, any user that cannot download the file within 36 hours aborts the download[3]. Finally, although BitTorrent does not explicitly provide incentives for seeding, most of the users altruistically stay some time after downloading. To realistically model this time period, we use the seeding time distribution of the peers in our microscopic crawlings, with an average of 3.37 hours (see Figure 22).

- *Speed distributions.* To realistically reflect the Internet access connection of the different users contained in our trace logs, we first associate each IP address with a country, using a freely available geolocation database [82]. Based on the country of origin, the Ookla database [121] provides us with the median down/uplink capacity of each user[4]. To see the characteristics of these speeds on a per-country level, we make reference to Figure 31.

Incentive Model

We have extended the existing *BitTorrent* implementation of the OctoSim simulator as described in Section 4.3.1. To this end we have also validated the correctness of the implementation by comparing the simulation results produced with live experiments (see Appendix A.4).

To model *FairSwarm.KOM*, we have accurately implemented the different protocol specifications presented in Chapter 5. Also, we have modified the existing BitTorrent implementation such that FairSwarm.KOM can operate on top of BitTorrent's content delivery and discovery mechanism. Details about the changes performed can be found in Section 5.7. Importantly, our simulation model of FairSwarm.KOM also considers the exchange of messages between the peers to establish a one-hop indirect reciprocation infrastructure. This allows us to study overhead aspects of FairSwarm.KOM.

To model the *Shared History* incentive mechanism, we utilise the FairSwarm.KOM implementation. However, instead of using one-hop information for decision making, every peer has global knowledge about all transactions occurred in the P2P network (without exchanging any network messages). This represents an upper bound with regard to a peer's knowledge database that can only be approximated in reality when all participants of the system would share their local experiences.

Finally, Table 14 gives an overview of important protocol parameters that we use in our evaluation. The peers' outdegree k as well as the peers' neighbourhood size is taken from the default setup of currently existing BitTorrent clients. As mentioned in the previous section, these are delivery-specific parameters that are optimised for mesh-based systems such as BitTorrent. To enable a fair comparison, we have simply adopted these values for all three incentive approaches. Since FairSwarm.KOM users exchange messages to share information about third-party behaviour, the table also contains the chosen parameter values as determined in previous section. Finally, FairSwarm.KOM as well as the Shared History approach employ our proposed default service policy for decision-making (cf. Section 5.6.1).

3 We find through simulations that 36 hours is enough time to get a download success ratio over 99% in the presence of seeders for all access links and file sizes used in our experiments.

4 Note that we refrain from using the bandwidth distribution of Isdal et al. [50] in this experiment, since this data does not contain a mapping from the geographical location of the peers to their corresponding download and upload speeds.

	BitTorrent	FairSwarm.KOM	Shared History
Incentive Details			
Incentive	bilateral	multilateral	multilateral
Ranking of peers	rate-based	volume-based	volume-based
Information horizon	local view	1-hop view	oracle view
Common parameters			
Peer outdegree k	5	5	5
Max. number of overlay neighbours	50	50	50
FairSwarm.KOM-specific parameters			
Percentage update threshold (in %)	–	30	–
Number of probabilistic coupons	–	40	–
Service Policy	–	default	default

Table 14: Overview of incentive mechanisms and parameter values used in our evaluation.

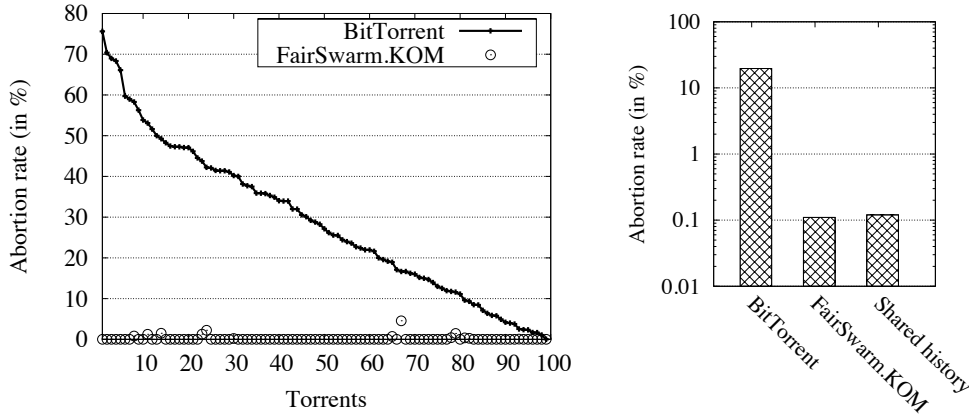
6.3.2 *Availability*

We next present the results of the competitive analysis of the different P2P incentive mechanisms under study. To be consistent with our solution space analysis, we report only the results obtained from one out of the five torrent subsets. Note that the results of the other torrent subsets are, however, very similar to those presented below.

We begin by studying the long-term availability of content achieved by each incentive approach. From a technical perspective, a file can be considered as unavailable if at least one of its chunks is not accessible within a swarm. As shown in Section 3.4.2, this often coincides with the lack of seeders. In seedless states, *(i)* users are typically not able to reconstruct a complete file and *(ii)* users' download rates rapidly drop to 0 KBps. Consequently, a prominent metric for measuring unavailability is the user-level *abortion rate*, as most users are only prepared to wait a limited length of time during an unavailability period. Recall that this time amounts to 36 hours at maximum in our experiments, which is a very conservative assumption compared to the impatience of users observed in reality (cf. Section 3.3.2).

Since one of our main objectives is to show that FairSwarm.KOM increases the availability properties of the BitTorrent system, we first compare the abortion rates of BitTorrent to FairSwarm.KOM. Figure 46a gives an overview of the obtained results on a per-torrent level. In particular, the x-axis of this figure sorts the torrent swarms in descending order, according to the abortion rates observed with BitTorrent. The y-axis, on the other hand, depicts the download abortion rate of a given torrent. In general, we can observe that, in BitTorrent, overlay swarms suffer heavily from file unavailability, with abortion rates amounting up to 74%. This observation is inline with the findings of our measurement study in the BitTorrent system. FairSwarm.KOM, however, seems to significantly improve file availability, as most of the torrents exhibit abortion rates of <1%, with 4.55% in the worst case.

One limitation of Figure 46a is that it clearly hides content popularity. That is, the torrent-wise representation does not reflect how many users have participated in the different torrents. Therefore, to extend this analysis, we also consider statistics on a system level. Figure 46b summarises the abortion rates of all users, irrespective of the torrents they have participated in (note that the y-axis is in log-scale). This time, we also incorporate the results obtained from the Shared History incentive mechanism. We see that, in BitTorrent, 19.67% of the users are unable to finish their downloads. This



(a) (b)

Figure 46: Overview of abortion rates on a per-torrent level (Subfigure (a)) and system level (Subfigure (b)).

clearly highlights that file unavailability is not only a problem of unpopular content, as in our simulation-based study every 5th BitTorrent user is affected by this problem. FairSwarm.KOM, in contrast, can reduce this significant abortion rate from 19.67% to 0.11%, thereby ensuring almost persistent file availability (>99%). Interestingly, the Shared History mechanism cannot even improve this figure. Here, we observe a download abortion rate of 0.12%.

6.3.3 *Performance*

So far we have validated that FairSwarm.KOM indeed improves file availability in the BitTorrent system. The next important step is to see whether this improvement comes at the expense of client download performance. This is important because we have observed this issue already in our analysis of the eMule-based tit-for-tat incentive. Specifically, this approach guaranteed file availability, but users experienced an extremely low perceived quality of service (cf. Section 4.3.3). Download performance is, however, highly important, as it determines user satisfaction and thus the success of the entire system. Once it becomes too low, users tend to abort downloads [59].

To quantify performance, we measure the download throughput of the users; this is defined by the total download amount divided by the download time. Similar to the previous analysis, we begin by inspecting the performance achieved by FairSwarm.KOM in comparison to BitTorrent. In this regard, Figure 47 gives an overview of the mean average download throughput achieved by both approaches on a per-torrent basis. In general, we see that FairSwarm.KOM clearly outperforms BitTorrent in the majority of overlay swarms. Specifically, the approach improves the mean average download performance of a given overlay swarm by more than 103%. That is, it decreases the download times on a per-torrent level by a factor of more than 2.

Clearly, the torrent-wise representation of download throughputs neglects content popularity and also hides performance constraints that could be caused by file unavailability. Therefore, Figure 48 also plots the cumulative distribution function of user download throughputs for BitTorrent, FairSwarm.KOM, and the Shared History mechanism. To compute this figure, we have considered all the users participating in our simulation-based study. From the depicted results, we can make three major observations. First, more than 20% of the BitTorrent users achieve a download throughput of less than 1 KBps. This coincides with the fraction of users that abort downloads, as observed in our availability analysis.

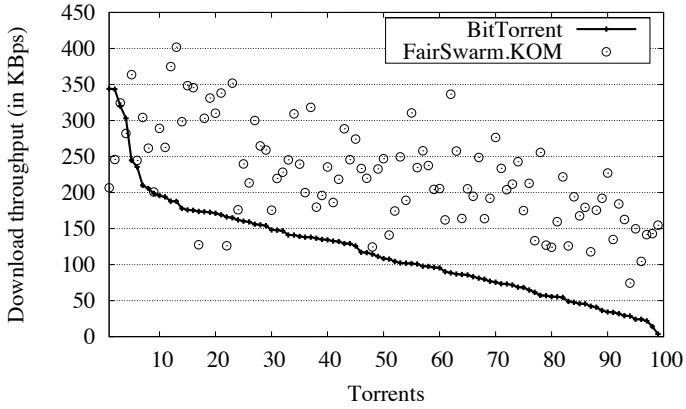


Figure 47: Overview of the mean average download throughput of BitTorrent and FairSwarm.KOM on a per-torrent level.

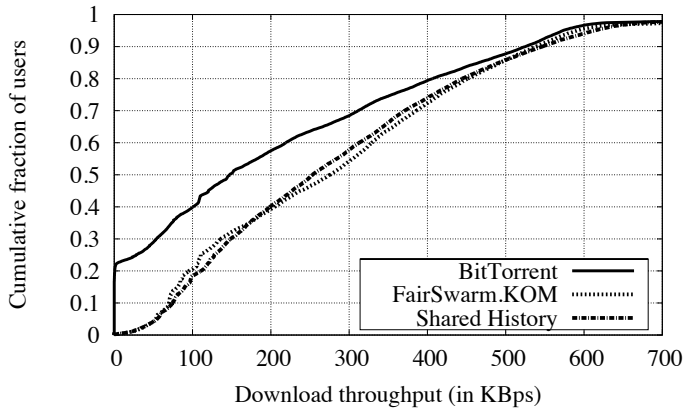


Figure 48: Overview of download throughputs of the different approaches (system-level).

Second, it can be also noted that FairSwarm.KOM vastly improves the download performance of BitTorrent users. More precisely, it increases the median download throughput of BitTorrent by more than 86%. Further, we find 43.3% of the users would gain a performance boost of more than 88% when switching from BitTorrent to FairSwarm.KOM in our workload under study. While these results are at first glance surprising, FairSwarm.KOM benefits from the significant increase in nodes (>19%) which now find content available. This allows users, which previously could not access the content, to download at high rates. Also, it frees up additional upload bandwidth that increases total system capacity.

Last, when comparing the throughput curves of FairSwarm.KOM and the Shared History approach, it can be noticed that both incentive mechanisms achieve a similar performance (cf. Figure 48). This means, even with global knowledge, the system would not perform better as it already does with one-hop reputation. An important issue, however, is whether FairSwarm.KOM achieves this high level of performance at the expense of fairness. We will further investigate this issue in the following.

6.3.4 *Fairness*

The previous subsections have studied the long-term effectiveness of FairSwarm.KOM with regard to file availability and download performance. In this regard, it has been found that FairSwarm.KOM offers superior performance and availability when compared to BitTorrent. Furthermore, these properties could not be improved by even

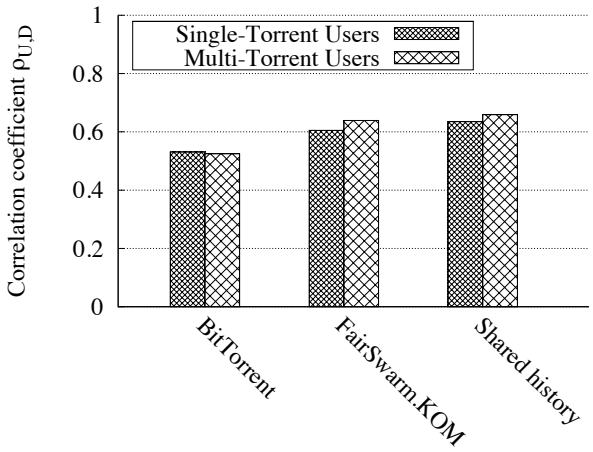


Figure 49: Overview of fairness properties of the different incentive mechanisms. Correlation coefficient $\rho_{U,D}$ quantifies to the relation between the users' provided upload rate (U) and their experienced download throughput (D).

the Shared History approach, indicating that FairSwarm.KOM is highly effective at disseminating the necessary information. Recent work has shown that high download performance can be subject to systematic unfairness [31]. The last step to take is, therefore, to analyse the fairness properties of the different approaches.

To quantify fairness, we make use of our definition presented in Section 4.3.2, which defines fairness by the amount of reciprocated data generated by contributions. This means, any peer x with an upload rate U_x should get a higher download throughput than any other peer y with an upload rate $U_y < U_x$. To measure this for the entire system, we compute the relation between the upload rate of users (U) and their experienced download throughput (D), which is represented by the correlation coefficient $\rho_{U,D}$. More precisely, we compute this coefficient for all users that join *(i)* just a single torrent and *(ii)* multiple torrents, as shown in Figure 49. Splitting users into these two groups allows us to quantify whether users persisting as file replicas (cross-torrent seeders) truly receive reciprocation for contributions in torrents that they have previously downloaded in.

In general, Figure 49 shows that all incentive mechanisms offer highly positive fairness properties. That is, there is a high correlation between the users' upload contribution and download rewards. This suggests a good service differentiation for the entire upload capacity spectrum. When comparing the different approaches to each other, we observe that FairSwarm.KOM even improves the fairness properties of BitTorrent. In particular, compared to BitTorrent, FairSwarm.KOM increases the fairness correlation coefficient $\rho_{U,D}$ by 19.29%. Importantly, this also holds for multi-torrent users, acting as file replicas in multiple overlay swarms. Furthermore, it can be also observed that even if users have global knowledge for decision making, the fairness properties of FairSwarm.KOM would only be improved further by 4%. In fact, these observations highlight that FairSwarm.KOM's reputation infrastructure provides sufficient information to the users such that they can recognise contributions of others participants across time and torrents.

To conclude, after inspecting the fairness properties of the different approaches, we can confidently say that FairSwarm.KOM's performance and availability improvements do not come at the expense of the individual users' fairness. In contrary, it has been shown that FairSwarm.KOM achieves fairness levels that are comparable to those of an incentive system using global knowledge of the network. Most importantly, FairSwarm.KOM achieves this by only incorporating the information of one-hop neighbours.

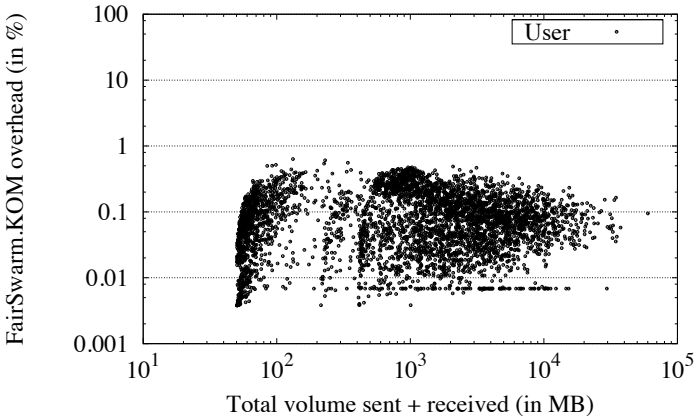


Figure 50: FairSwarm.KOM communication overhead in comparison to the total amount of data sent and received.

6.3.5 *Overhead*

So far it has been shown that FairSwarm.KOM has outperformed BitTorrent in all metrics under consideration. The last step we take is to inspect the overhead of FairSwarm.KOM. As discussed in the previous chapter, one of FairSwarm.KOM's design principles is to propagate reputation information to at most one-hop in the overlay network, both to promote scalability and robustness. While the superior simulation results of FairSwarm.KOM already indicate that FairSwarm.KOM's signalling traffic is not the dominating factor in network communication, it is still important to understand how significant overhead is in comparison to the total amount of data exchanged [47]. Furthermore, FairSwarm.KOM's data exchange specification requires that data chunks are symmetrically encrypted when being transmitted, inducing an additional computational overhead. To this end, other approaches such as network coding have already failed deployability in P2P systems, particularly because of their computational complexity [40, 124]. Thus, we also consider this as an important overhead metric to study. We briefly focus on these two overhead aspects in the following.

Communication Overhead

BitTorrent's incentive mechanism solely relies on direct observations and local histories. Thus, the approach does not introduce any communication overhead. In FairSwarm.KOM, however, peers must exchange messages about third-party behaviour. To quantify how much this control traffic contributes to the total network traffic, we divide the aggregate amount of data sent and received due to FairSwarm.KOM traffic by the total amount of data sent and received at a given node. Figure 50 summarises this percentage overhead for every user in our simulation-based study, alongside the total data volume sent and received. Note that both axis are in log-scale.

In general, the figure shows that the communication overhead of FairSwarm.KOM is very low, compared to the overall traffic caused by data exchanges. Specifically, on average, FairSwarm.KOM's control traffic accounts only for 0.11% of the total traffic. As can be also seen, this is irrespective of how much users upload and download, highlighting the scalability of the approach. Finally, even in the worst case, this overhead figure never exceeds 0.63% for a user.

In fact, although the percentage overhead is extremely low, it can still be a value of note when considering large files. However, even with the added signalling traffic of FairSwarm.KOM, the users still yield a median performance improvement of more than 86% compared to BitTorrent, making this bandwidth investment worthwhile.

	FairSwarm.KOM operation (data provider)	Time
	CPU-centric operation	
1	Encrypt chunk (256 KB)	1.45 ms
2	Hash encrypted chunk (256 KB)	0.92 ms
3	Hash coupon	0.08 ms
4	Hash secret key	0.01 ms
5	Signature coupon (RSA)	1.78 ms
6	Signature transaction label + secret Key (RSA)	1.98 ms
7	Verification of consumer's signature (RSA)	0.12 ms
8	Asymmetric encryption of secret key (RSA)	0.18 ms
		= 6.52 ms
	Communication operation	
10	Data transmission (256KB chunk)	20,480.00 ms

Table 15: Timing of cryptographic and sending operations, assuming an average per slot bandwidth of 100 Kbps. Timings for operations 1-8 are determined using the average value of 1,000 executions.

Moreover, if really necessary, overhead can be adapted (i.e., further reduced) through parameter modification based on different environments (cf. Section 6.2).

Computational Overhead

In FairSwarm.KOM, peers have to perform cryptographic operations during the exchange of data (see Section 5.5). While a data consumer must only prepare its signature attesting the receipt of data, the provider must perform a few operations that can delay the sending of user data. Table 15 presents a listing of all operations that must be performed at the provider's site (operation 1-8). For each of these operations, the table also lists the CPU time needed when performing this operation using the Java cryptography architecture FlexiProvider[5]. Specifically, to measure these times, we have used a dual core Pentium 2.4 GHZ with 2 GB RAM and assumed a chunk size of 256 KB. This figure represents the average chunk size observed in our BitTorrent measurement study. We see from the table that the different cryptographic operations are highly efficient, as only symmetric cryptography is employed to encrypt/decrypt the data to be transferred. Added up, we obtain a CPU-time of 6.52 milliseconds per data transaction.

We now contrast this time against the time needed for transferring a data chunk of 256 KB. Let us assume the following without the loss of generality: the typical outdegree k of a user in BitTorrent/FairSwarm.KOM is 5 [21]. Further, most ISPs offer upstream bandwidths of 500 Kbps or less [25]. Thus, per upload slot, a peer usually has 100 Kbps upstream capacity available. When assuming that the bottleneck between a data provider and consumer is upstream capacity, the sending process of a 256 KB chunk will last 20.48 seconds. Consequently, the computational overhead introduced by FairSwarm.KOM reduces the uplink utilisation of a peer by only 0.03%, which is negligible.

In summary, the results show that FairSwarm.KOM's communication and computational overheads are marginal and by all means justifiable, compared to the improvements that this incentive mechanism offers to the tit-for-tat incentive scheme of BitTorrent.

5 http://www.flexiprovider.de

6.4 ROBUSTNESS TO UNTRUTHFUL USER BEHAVIOUR

The previous section has validated that indirect reciprocation is vital for overcoming the limitations of current P2P content distribution systems, particularly BitTorrent. However, the use of information that goes beyond direct observations and local histories (i.e., experiences of third-parties) has a significant disadvantage; it greatly increases the range of potential attacks possible by strategic peers. In this section, we study FairSwarm.KOM's robustness against the most common of these threats.

In the following, we begin by stating our methodology, while presenting the threat models we consider. Subsequently, for each individual threat, we report the results obtained and conclusions drawn.

6.4.1 *Experimental Methodology*

When analysing the vulnerability of FairSwarm.KOM to strategic or malicious attacks, it is important to study the impact of those threats in a large variety of overlay swarms. For example, it is intuitive that torrents predominantly consisting of seeders may favour free-loading, because these kinds of swarms offer enough excess capacity to be exploited by free-riders. Also, sparsely populated swarms further ease the exploitation of this source of altruism, because non-contributors must only compete with a few other participants. On the other hand, when a swarm is dominated by leechers, the situation may be reversed to the disadvantage of the attacker. From this, we can deduce that the success of a given threat depends on the characteristics of the swarm it is performed in. Thus, when studying only a single torrent configuration, one will gain only a very limited understanding of the effectiveness of different attacks in the wild.

To remedy this issue, we make use of our macroscopic measurement data and opt for trace-based simulations. Specifically, this data allows us to reconstruct a wealth of overlay swarms as they exist in the Internet today. These swarms are then taken as input for our experiments to simulate different threats to FairSwarm.KOM. As in all simulations before, we use for our experiments the OctoSim simulator, with all the modifications presented in Section 4.3.1.

Within this subsection, we first present the different threat models that we consider in our analysis. Subsequently, we detail our approach to generate a large set of overlay swarms in which the different attacks are carried out. Finally, we describe the characteristics of attackers and control nodes operating in these swarms.

Threat Models

For the purpose of our analysis, we assume that selfish/malicious nodes can misbehave as follows in FairSwarm.KOM. First, a peer may attempt to download content without contributing anything to the system. In particular, this free-loading can be performed by maintaining the peer's identity (ordinary free-riding) or by continuously changing its identity (whitewashing). Importantly, since FairSwarm.KOM users can generate identities arbitrarily and at will without being charged for this, both attacks appear likely. Furthermore, a peer may attempt to boost its standing in the system by creating multiple FairSwarm.KOM accounts, each attesting data transactions that have not occurred in reality. This can be performed with Sybil nodes that maintain their identities and participate in the data exchange process (Sybil attack with fixed identities). Or, alternatively, it can be run with Sybil nodes that change their identities as soon as they have consumed a pre-defined amount of data (Sybil attack with changing identities)[6].

6 Note that we refrain from studying collusion between multiple peers, as this threat is implicitly covered by the Sybil attacks under study.

Parameter	Percentile				
	5th	25th	50th	75th	95th
File size (in MB)	2	45	240	733	1,440
Number of online leechers	2	5	10	33	998
Seeder-to-leecher ratio	0 (0.01)	0.32	0.93	2.38	8.6

Table 16: Overview of simulation input parameters. Note that the seeder-to-leecher ratio below the 17th percentile is 0. As content is unavailable in those circumstances, we use the 17th percentile (=0.01) as most-left value instead.

Workload Model

To generate a wealth of lifelike overlay swarms, we make use of our macroscopic data set consisting of trace logs from more than 46,000 BitTorrent swarms. Through this, we are able to compute the cumulative distribution function of the ratios between seeders and leechers, content sizes, and the absolute number of downloaders/seeders. From each of these distributions, we have taken the 5th, 25th, 50th, 75th and 95th percentile, which are shown in Table 16. This allows us to preserve the most important characteristics of currently existing overlay swarms. For instance, as can be seen from the table, skewness is high in all distributions, which is common for P2P environments.

Each of these values is then combined by fixing one parameter while varying the values from the others. This results in 125 different torrent configurations. To have a representative client bandwidth distribution in these overlay swarms, we assign the Internet access links of the users according to the measurement data of Isdal et al. [50]. Since the resulting access link distribution of each torrent depends on the chosen random seed, we have used 20 different seed values. All things considered, our final workload consists of $(20 \cdot 125) = 2{,}500$ characteristic overlay swarms, that we use for our study.

Attackers and Control Nodes

To understand the worst-case scenario, we provision attackers with infinite download capacity. This is a reasonable assumption because most of the Internet access connections today are highly asymmetric with downstream capacities that are often a multiple of the upstream capacity (cf. Figure 14). Furthermore, in fraud attempts where available upstream capacity becomes important, we consider three different types of attackers: a low capacity peer (LCP), a medium capacity peer (MCP), and a high capacity peer (HCP), each endowed with upstream capacities of 512, 1,024, and 4,096 Kbps, respectively. The rationale behind this is to evaluate the effectiveness of a given threat with regard to the wide-spectrum of currently available Internet access links [25].

For comparability reasons, in each experiment, we always bring up three cooperative control nodes that *(i)* simultaneously join the swarm with the node(s) performing a given attack and *(ii)* also possess the same upload capacity as the attackers. As such, these control nodes act as reference point and reflect the performance that can be achieved when users adhere to FairSwarm.KOM's protocol specification.

6.4.2 *Free-Riding and Whitewashing*

We begin by studying the vulnerability of FairSwarm.KOM to free-riding. This includes ordinary free-riding as observed in many currently deployed P2P content distribution system (e.g., BitTorrent, eMule, and Gnutella), and free-riding with the change of

Parameter	Value
Simulated environment	
Overlay swarms considered	2,500 (cf. Table 16)
Swarm population	Either FairSwarm.KOM or BitTorrent users
Client bandwidth distribution	Trace-based [50]
Attacker types	Free-rider, whitewasher
Attacker bandwidth (down-/upstream)	∞/0 Kbps
FairSwarm.KOM-specific parameters	
Number of probabilistic coupons	40
Percentage threshold (in %)	30
Peer outdegree k	5
Maximum number of overlay neighbours	50
Service policy	default
Attacker-specific parameters	
Identity change of whitewasher	After each downloaded chunk

Table 17: Overview of experiments: Free-riding and whitewashing.

identities, also called whitewashing. The latter threat is only relevant for incentive schemes relying on long-term incentives, e.g., FairSwarm.KOM. To this end, an attacker attempts to escape the consequences of its past by re-joining the system as a legitimate newcomer.

To study and quantify FairSwarm.KOM's robustness to both kinds of attack, we have performed three types of experiments. In the first experiment, we monitored the performance of an ordinary free-rider when downloading content in the 2,500 characteristic overlay swarms. These overlay swarms consist solely of FairSwarm.KOM users, using the protocol settings depicted in Table 17. The second experiment repeats the previous one, but measures the performance of a whitewasher that continuously changes its identity upon downloading a single file chunk. Finally, to have a baseline for the results, we also monitor the performance of an ordinary free-rider when the overlay swarms solely consist of BitTorrent users employing the tit-for-tat incentive strategy.

Table 18 summarises the download throughputs of the attacker nodes in BitTorrent and FairSwarm.KOM. For comparability reasons, the table also contains the throughputs achieved by the three different types of control nodes. We start with considering the performance of ordinary free-riders in BitTorrent and FairSwarm.KOM. It should be noted that free-loading is highly effective in the current BitTorrent system; a free-rider achieves almost the same download performance as a low provisioned node, contributing with a non-negligible amount of upstream bandwidth (512 Kbps). Worse, this picture does not even change for an extremely well-resourced node, contributing 4,096 Kbps upstream capacity. In particular, compared to a free-riding node, the high capacity peer achieves a performance improvement of only 16.69 %.

On the other hand, Table 18 shows that FairSwarm.KOM does also not prohibit free-riding. That is, the free-riding node achieves a mean average download throughput of 124 KBps and eventually finishes its download in all overlay swarms under consideration. The latter is, however, expected as our default service policy refrains from using contribution thresholds that peers must exceed to download content. This is a necessity to enable new nodes to join in the system and to improve overall system throughput (cf.

User Type (upstream bandwidth)	BitTorrent Speed (KBps)	FairSwarm.KOM Speed (KBps)
Attackers:		
Free-rider (0 Kbps)	877.76	124.08
Whitewasher (0 Kbps)	(–)	230.45
Control nodes:		
Low capacity peer (512 Kbps)	885.15	652.52
Medium capacity peer (1,024 Kbps)	931.33	804.17
High capacity peer (4,096 Kbps)	1,024.08	1,034.81

Table 18: Overview of performance of free-riders, whitewashers, and three cooperative control nodes, when downloading content in 2,500 characteristic overlay swarms. All figures are average values, computed over all swarms.

Section 5.6). Nevertheless, compared to BitTorrent, FairSwarm.KOM's default service policy highly limits the opportunity for effective free-loading, as non-contributors achieve only 11-19% of the performance of obedient nodes.

The reason for FairSwarm.KOM's increased robustness to free-loading must be clearly attributed to the use of one-hop information for decision making. In particular, FairSwarm.KOM users (particularly seeders) have the opportunity to evaluate whether leechers reciprocate to other users in the system. This means, in order to obtain a downloading slot, leechers must prove their cooperativeness by forwarding content to other participants. In bilateral incentive schemes (i.e., tit-for-tat), however, seeders cannot see this and can thus easily be exploited by free-riders.

To demonstrate this, we group the different torrents by their seeder-to-leecher ratio, and, for each category obtained, we compute the users' average download throughput. Figure 51 shows these results for BitTorrent and FairSwarm.KOM. In fact, it can be observed that in overlay swarms with seeder-to-leecher ratios greater than 0.32 (more than 70% of all existing torrents), free-riders can easily achieve a similar download performance than the highest contributing BitTorrent node (cf. Figure 51a). In FairSwarm.KOM, however, the free-rider's performance increases only slightly with increasing seeder-to-leecher ratios (cf. Figure 51b), clearly confirming our hypothesis.

Last, in FairSwarm.KOM, a free-rider could also attempt to whitewash its account by re-joining the system with different identities. In fact, our trace-based simulation analysis has revealed that this attack improves the performance of an ordinary free-rider by 88%. However, as shown in Table 18, compared to the cooperative control peers, these users also fail to achieve superior download performance, primarily because they can never generate a positive account balance. Consequently, both leechers and seeders rarely select these peers for uploading.

In essence, our analysis has revealed that FairSwarm.KOM is unable to *entirely* block free-riding because it refrains from using contribution thresholds that peers must exceed to download content. This design choice is, however, intentional in favour of total system throughput and to bootstrap cooperation (see Section 5.6). On the other hand, it must be also noted that FairSwarm.KOM significantly limits the effectiveness of free-riding in comparison to current tit-for-tat incentive strategies. Importantly, this has been confirmed in the entire range of currently existing overlay swarms characteristics. Last, due to FairSwarm.KOM's selective uploading strategy, the effectiveness of whitewashing is also clearly mitigated, even if not blocked. However, as mentioned before, the random contributions from which these users benefit are necessary in P2P content distribution systems. This is because obtaining information

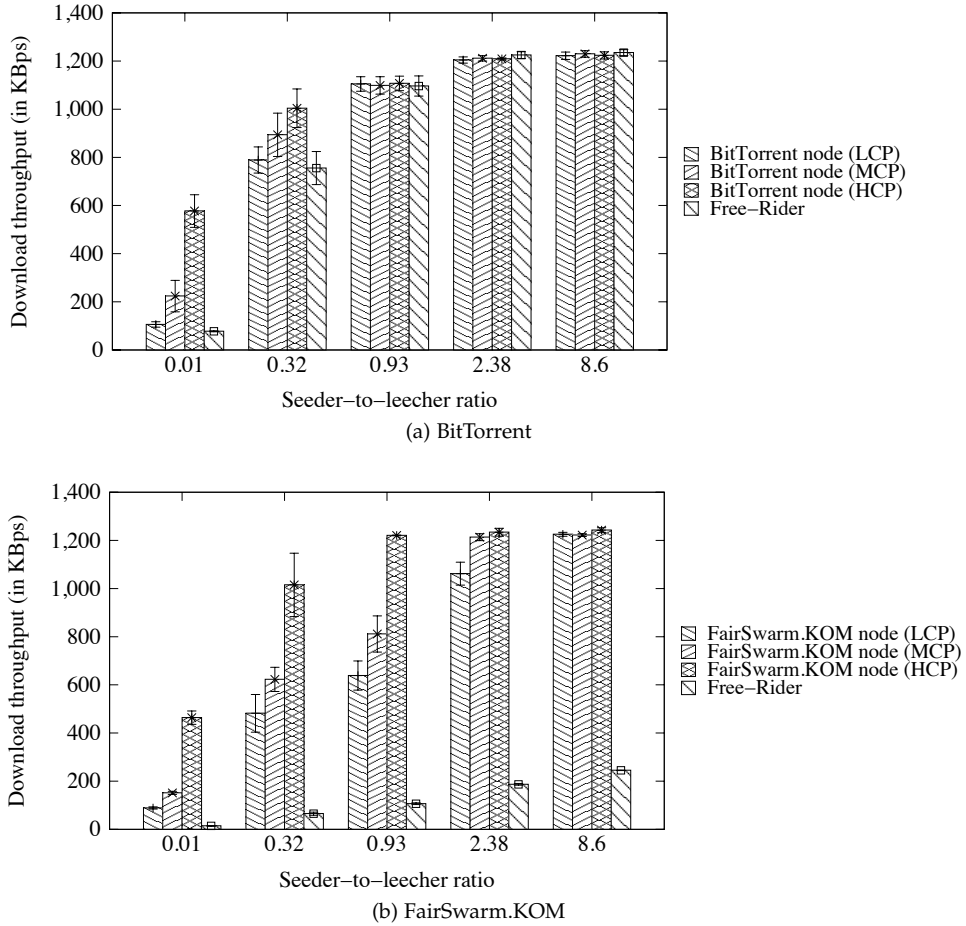


Figure 51: Download performance of three control nodes and free-riders for different seeder-to-leecher ratios. Error bars correspond to 95% confidence intervals.

about the cooperativeness of users requires contributing to all peers in the system, beneficial or not.

6.4.3 *Sybil Attack with Fixed Identities*

In FairSwarm.KOM, a more sophisticated attack could be that multiple strategising nodes cooperate to defraud the system, or, alternatively, a single node generates multiple identities to attest contributions that have never occurred. To see how this attack could work, consider Figure 52. Here, the strategising node has two FairSwarm.KOM accounts: its main account represented by node A and its Sybil account given by node S. Peer A is ordinarily participating in the download process (i.e., it downloads and uploads), while node S solely uploads content to other peers in the system. The intuition behind this is that both peers establish one-hop relationships to obedient FairSwarm.KOM nodes, so that peer A can claim back direct contributions as well as indirect contributions (via node S) from a common intermediate node I. To enable this, the attacker illegally issues and periodically updates a coupon $\zeta_{A,S}$, attesting fake contributions from peer A to S. This coupon is then submitted to all common intermediate nodes to improve peer A's standing in the system.

Clearly, the recipients of coupon $\zeta_{A,S}$ could easily detect this attack by keeping track of the amount of data that has been exchanged between two peers. For instance, it appears unrealistic that A contributes to S one gigabyte of data within a minute. Further, abnormally high account balances that deviate significantly from the accounts of others

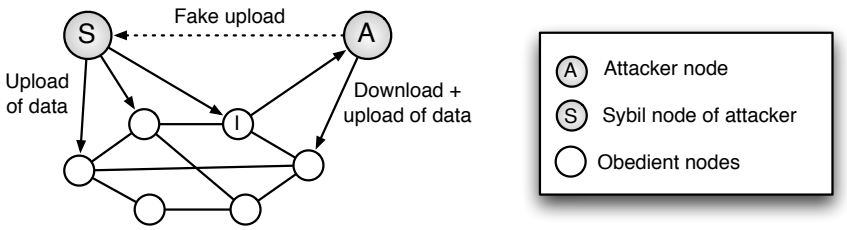


Figure 52: Illustration of a Sybil attack with fixed identities in FairSwarm.KOM.

Parameter	Value
Simulated environment	
Simulated overlay swarms	2,500 (cf. Table 16)
Overlay swarm population	FairSwarm.KOM users
Client bandwidth distribution	Trace-based [50]
Attacker types	low capacity peer (LCP), medium capacity peer (MCP), high capacity peer HCP)
Attacker bandwidth (down-/upstream)	LCP (∞/512 Kbps), MCP (∞/1,024 Kbps), HCP (∞/4,096 Kbps)
FairSwarm.KOM parameters	
Number of probabilistic coupons	40
Percentage threshold (in %)	30
Peer outdegree k	5
Maximum number of overlay neighbours	50
Service policy	default (without MaxFlow), default

Table 19: Overview of experiments: Sybil attacks with fixed identities.

may be also suspicious. On the other hand, the attacker could simply circumvent this check as follows. Every t minutes, it increases the data field of coupon $\zeta_{A,S}$ according to a data volume V that is proportional to the peer's upload capacity, e.g., double of its upload capacity U. Volume V can thus simply be computed by $V = t \cdot U \cdot 2$. When doing so, the strategising node will be only hardly distinguishable from other nodes in the system.

To study the effectiveness of the above attack, we have performed two experiments. In both experiments, three types of attackers (solely differing in their upstream capacity) attempt to download content in the 2,500 characteristic overlay swarms. All participants in these swarms are FairSwarm.KOM users, using the protocol settings illustrated in Table 19. In the first experiment, however, FairSwarm.KOM users refrain from using the max-flow min-cut theorem to detect Sybil attacks (cf. Section 5.6.1). In the second experiment, on the other hand, these users follow our recommendation and use the MaxFlow algorithm. Further, in both experiments, we set $t = 2$ minutes and assume that a strategising user allocates a fifth of its available upload capacity to node S[7]. The latter is necessary to establish one-hop relationships between a victim I and the attacker accounts A and S. Otherwise, victim I would not accept the coupon $\zeta_{A,S}$ gossiped by node A.

Figure 53 summarises the mean average download throughput for the different attackers types, either when FairSwarm.KOM users are using our proposed counter-

7 We have also experimented with smaller/higher time intervals and different bandwidth allocations, but the conclusions drawn are similar to those presented here.

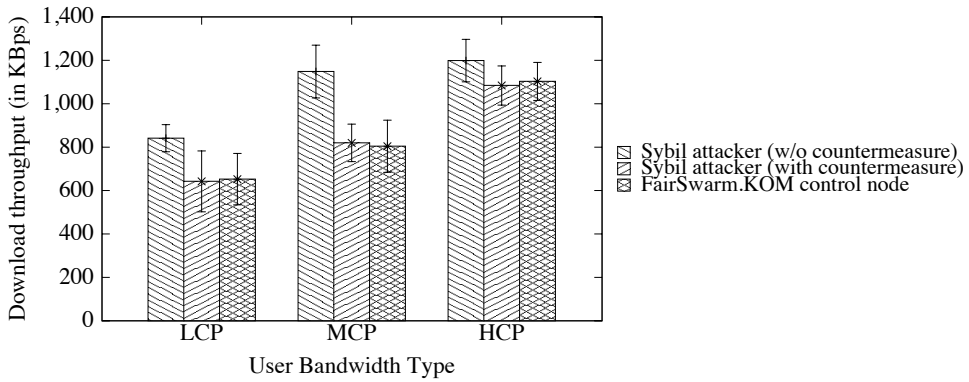


Figure 53: Download throughput of Sybil attackers in FairSwarm.KOM. Without (w/o) and with countermeasure means that FairSwarm.KOM users either refrain from using or use the min-cut max-flow theorem to protect themselves against Sybil attacks.

measure to Sybil attacks or when they do not. To benchmark the performance achieved, the figure also shows the download throughputs of the corresponding control nodes.

The results of Figure 53 clearly suggest that a Sybil attacker can achieve superior download performance in FairSwarm.KOM, when users refrain from using the MaxFlow algorithm (w/o countermeasure). Specifically, the figure shows that the attacker's download throughput improves by 28-42%, compared to an obedient control node originating from the same bandwidth class. In fact, due to this significant gain in performance, this threat is therefore likely to become adopted by other selfish users.

On the other hand, when using FairSwarm.KOM's default service policy (with countermeasure), the impact of this attack is *effectively* blocked. As noticeable in Figure 53, the download throughput of the attack nodes are almost identical to those of the corresponding control nodes. Concretely, the medium capacity attacker performs 1.8% better, while the low and high capacity attacker performs 1.1-1.6% worse. This proves that, through the aid of the min-cut max-flow theorem, a Sybil attacker can only claim back contributions that it has truly provided to other nodes in the system.

Also, as already discussed in Section 5.6.1, it is important to recognise that this situation will not change for an attacker, if it creates additional Sybil nodes, e.g., S', S'', and so on. This is because the attacker must always allocate a fraction of its total upstream capacity to these Sybil nodes, in order to establish one-hop relationships. However, when applying the MaxFlow algorithm, the sum of contributions achieved by the Sybil nodes is always bounded by the attacker's total upload capacity. In other words, this is equivalent to the case when serving content with full capacity of service to other participants using a single identity.

To conclude, from the point of view of selfish users, the Sybil attack as described above is not beneficial in FairSwarm.KOM. This is because the attackers have to undertake an additional effort to realise such an attack. In particular, they must maintain multiple identities, introducing an additional communication overhead for the gossiping of illegally created coupons, without achieving any performance gain.

6.4.4 *Sybil Attack with Changing Identities*

So far we have studied the effectiveness of selfish/malicious users that are reluctant to contribute upload resources. In addition to this, we have also considered a Sybil attack where an attacker A creates a single identity (or alternatively multiple ones) attesting fake contributions to it. By doing so, the attacker attempts to claim back contributions using one-hop indirect reciprocation. However, alternatively, the attacker could also approach the attack from the opposite way round. That is, instead of requesting content

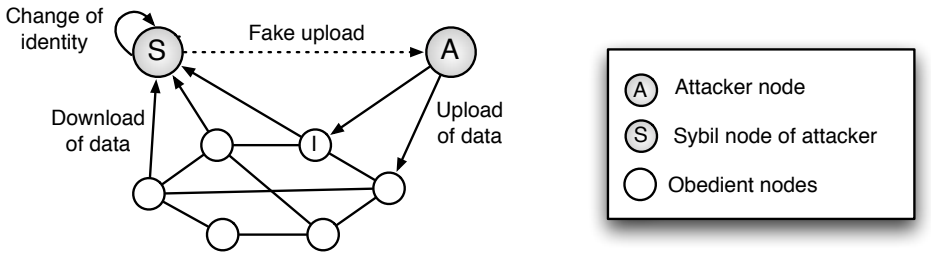


Figure 54: Illustration of a Sybil attack with changing identities in FairSwarm.KOM.

Parameter	Value
Simulated environment	
Simulated overlay swarms	2,500 (cf. Table 16)
Overlay swarm population	FairSwarm.KOM users
Client bandwidth distribution	Trace-based [50]
Attacker types	low capacity peer (LCP), medium capacity peer (MCP), high capacity peer HCP)
Attacker bandwidth (down-/upstream)	LCP (∞/512 Kbps), MCP (∞/1,024 Kbps), HCP (∞/4,096 Kbps)
FairSwarm.KOM parameters	
Number of probabilistic coupons	40
Percentage threshold (in %)	30
Peer outdegree k	5
Maximum number of overlay neighbours	50
Service policy	default
Attacker-specific parameters	
Identity change after P% of file download	10%, 20%, 50%

Table 20: Overview of experiments: Sybil attacks with changing identities.

by itself, the attacker could give this responsibility to a Sybil node that continuously changes its identity.

To see how this attack could work with changing identities, consider Figure 54. Assume that attacker node A and Sybil node S are able to simultaneously access a shared folder. For instance, if the attack is carried out from a single physical machine, this is can be implemented without any problems. In this scenario, the sole responsibility of node A is then to serve other participants with already downloaded content. Importantly, to yield the maximum profit, this must be done with the full service capacity. The reason behind is to create as many outgoing contribution edges as possible in the network, so as to boost peer A's standing in the system. Sybil node S, on the other hand, attempts to exploit the continuously increasing account surplus of peer A. To do so, both nodes illegally issue and periodically update a coupon $\zeta_{A,S}$, attesting fake contributions from peer S to A. Node S then gossips this coupon to common intermediate nodes (e.g., node I in Figure 54), to simultaneously request content from these nodes. This attack is also likely to work in practise, because node S pretends to have uploaded to node A, and A has truly uploaded to intermediate peer I ($S \rightarrow A \rightarrow I$). Thus, peer S is indeed allowed to claim back contributions via one-level of indirection from node I.

There are a few things that limit the effectiveness of such a sophisticated attack. For one, the above attack can only be launched against leechers in FairSwarm.KOM. This is because peer A cannot upload to seeders, and, therefore, one-hop relationships cannot be established with these users. Furthermore, the effectiveness of the attack against leechers depends on three points: *(i)* peer A's upload speed, *(ii)* peer S's success at establishing an overlay connection to leechers that are consuming from peer A, and *(iii)* peer S's account balance compared to other leechers in the system. The first issue clearly depends on peer A's physical upstream bandwidth. The second point, however, becomes challenging for large swarm sizes (> 50 peers) in which users could have already exceeded their maximum neighbourhood size, which is by default 50[8]. Finally, the third point needs a closer inspection. It appears intuitive that the more content peer S downloads, the lower is its chance to win a download slot among other competing leechers. Thus, a logical consequence would be to continuously change peer S's identity to best exploit peer A's increasing standing in the system. Frequent changes of identities have, however, the consequence that Sybil node S may be unable to reconnect to victim I, since other users have already occupied its neighbourhood place. Also, content requests from the same IP-address, but with a frequently changing long-term identifier can be easily detected and blocked.

To address this, we make the following assumptions. First, we assume that every $t = 2$ minutes, both nodes S and A illegally increase the corresponding data field of coupon $\zeta_{A,S}$ by data volume V, thereby attesting fake contributions from S to A. As in our previous experiments; the exact data volume is given $V = t \cdot U \cdot 2$, whereas U is the attackers upload capacity[9]. Further, we assume that Sybil node S changes its identity as soon as it has downloaded another P percent of the file, where P is varied in our experiments.

To study the effectiveness of the Sybil attack with changing identities, we have monitored the download performance of three types of attackers when attempting to download content in the 2,500 characteristic overlay swarms. All participants in these swarms were FairSwarm.KOM users, using the protocol settings illustrated in Table 20. Furthermore, we have repeated these experiments for P=10%, 20%, and 50%. Figure 55 summarises the average download throughputs observed. When comparing the results of attackers and control nodes, we clearly observe that the attackers achieve a significantly lower performance. In particular, depending on the chosen parameter value P and the attacker's upload capacity, the observed performance loss is between 59-75%. Considering that these attackers serve content with the same capacity as our control nodes, the performance degradation is significant.

After a closer inspection of the results obtained, we find that the attack fails to succeed, mainly because the attackers lose their opportunity to download content from seeders. That is, since they cannot demonstrate their cooperativeness to seeders, obedient FairSwarm.KOM users are always preferred in the seeders' upload queues. This situation only changes when all other (obedient) users have finished their downloads and enough excess capacity is available that can be ultimately tapped by the attackers. Apparently, this is highly detrimental for download performance of the attack nodes.

We would like to add that we have also experimented with allocating a small fraction (i.e., a fifth) of node A's upload capacity to Sybil node S. The intuition behind it was to see if this improves performance, since in this case, Sybil node S is also contributing to the system, potentially improving its chances to obtain downloading slots from seeders. However, we observed no significant change in download performance, which must be attributed to the combination of two things. First, since node A's available upload capacity is decreased, its standing in the system increases more slowly. Second,

8 This is further exacerbated by the fact that users quickly drop overlay connections to peers that are solely downloading without contributing back. We neglect this fact, however, for the purpose of this analysis.

9 We also have experimented with other values of t, but observed no significant deviations between the results.

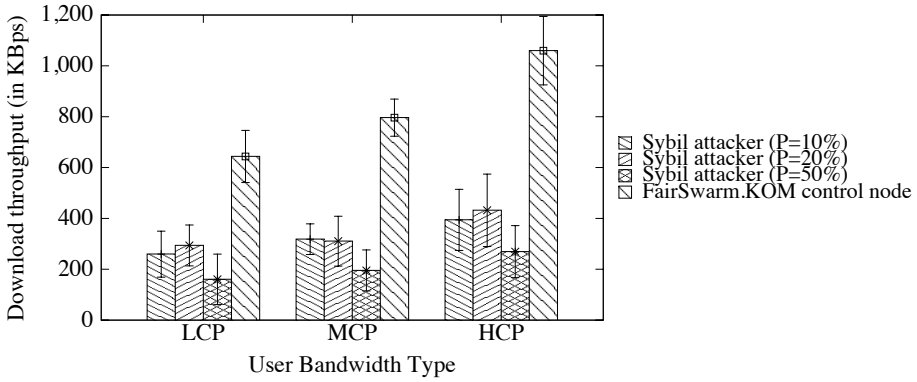


Figure 55: Download throughput of Sybil attackers with changing identities in FairSwarm.KOM. Every time P percent of the file is downloaded, the attacker changes the identity of node S (cf. Figure 54).

the comparatively less contribution of peer S cannot outweigh the node's resource consumption, so that it is only seldom favoured by seeders.

In essence, it has been shown that the discriminative uploading of FairSwarm.KOM's default service policy in combination with the peers' one-hop basis for decision making clearly mitigates the effectiveness of Sybil attacks with changing identities. In particular, for the same amount of upload contribution, the attacker nodes achieve a 59-75% lower performance than obedient nodes adhering to FairSwarm.KOM's protocol specification.

6.5 EVALUATIVE SUMMARY

This chapter has provided a quantitative evaluation of FairSwarm.KOM with respect to different properties. This section now revisits our evaluative aims to ascertain how well they have been fulfilled (cf. Section 6.1.1).

Our first evaluative aim was *to measure and quantify the potential performance and availability benefits that FairSwarm.KOM can offer BitTorrent*. This aim has been addressed through the use of up-to-date trace-based simulations, taking our month long measurement data of the BitTorrent system as input. Most notably, it was found that:

- FairSwarm.KOM provides superior performance compared to BitTorrent's tit-for-tat incentive. Specifically, the approach increases the mean average download throughput of overlay swarms by more than 103%. This improves the download rates of the users by 86.38% on median.
- FairSwarm.KOM reduces the abortion rate of users in the BitTorrent system from 19.67% to 0.11%, thereby guaranteeing almost persistent file availability (>99%).

Our second evaluative aim was *to ascertain if FairSwarm.KOM can improve BitTorrent's performance and availability without damaging user and system-wide fairness*. The trace-based simulations have revealed that FairSwarm.KOM provides high levels of fairness. In particular, it improves the already existing fairness properties of BitTorrent's tit-for-tat incentive mechanism by 19.23%, thereby achieving service differentiation that is proportional to the users' upload contribution. Thus, the newly devised incentive approach offers improved performance and availability properties, without hurting user and system-wide fairness.

Our third evaluative aim was *to measure the overheads related to utilising FairSwarm.KOM under current environments and workloads*. This aim has been also addressed through the use of trace-based simulations to find that the signalling traffic of FairSwarm.KOM accounts for only 0.11% of a user's total traffic on average, and for 0.63% in the worst

case. Importantly, this percentage overhead remains constant, irrespective of how much users download and upload, highlighting the scalability of the approach. Furthermore, it was also shown that FairSwarm.KOM's computation costs for cryptographic operations are negligible and do not introduce any significant delay that could impair a peer's uplink utilisation. To conclude, the overheads of FairSwarm.KOM are marginal and by all means justifiable, compared to the improvements that this mechanism offers to the tit-for-tat incentive scheme of BitTorrent.

Finally, our fourth evaluative aim was *to validate that FairSwarm.KOM is robust against common P2P attacks and to ensure that malicious users do not achieve superior performance to those that adhere to FairSwarm.KOM's policies*. This analysis is of particular importance, because relying on third-party experiences clearly increases the vulnerability of the approach to users attempting to strategise. To reasonably address this aim, extensive trace-based simulations of a large variety of existing overlay swarms have been performed. It was found that FairSwarm.KOM significantly limits the opportunity for effective free-riding. Moreover, it even offers increased robustness properties to this threat, when compared to current tit-for-tat incentive schemes. Specifically, while a free-rider in BitTorrent achieves almost the same download performance than a cooperative peer contributing with a non-negligible amount of upstream bandwidth (e.g., 512 Kbps), FairSwarm.KOM reduces the download rates of such non-contributors to only 11-19% of the performance of the contributing nodes. Despite this, it was shown that whitewashing and Sybil attacks are also highly mitigated in FairSwarm.KOM. This is because these attackers must undertake a non-trivial effort to realise such fraud attempts, but do not achieve any performance gains compared to users adhering to the protocol specification. In essence, these results clearly demonstrate FairSwarm.KOM's robustness to common attacks of strategising users, thereby meeting our last evaluative aim and completing our evaluation of FairSwarm.KOM.

With the evaluation provided in this chapter, we have also successfully addressed the third research goal of this thesis (cf. Section 1.4) and quantitatively confirmed that our devised solution meets the solution requirements posed in Section 4.4. Finally, a notable and interesting finding that is worth to be mentioned is that even when users have global knowledge for decision making, the results achieved by FairSwarm.KOM would not further improve. In more detail, our trace-based simulations have revealed that the performance, availability, and fairness properties achieved by the oracle-based incentive mechanism are close to what FairSwarm.KOM achieves by using only one-hop reputations. Thus, for the current BitTorrent system, the usage of complex virtual currency-based approaches to improve these properties cannot be justified.

7 CONCLUSION

This chapter concludes the dissertation. First, we briefly summarise the content and main findings of this thesis. Following this, the major contributions of the thesis are detailed, alongside other significant results. Subsequently, we revisit our research goals to ascertain how effectively they have been fulfilled. Last, we give an outlook on concrete future work opened up by the research performed in this thesis.

7.1 SUMMARY OF THESIS

Peer-to-peer content distribution offers many advantageous characteristics compared to the traditional client-server model for both content providers and end-users alike. Existing P2P systems, however, suffer from serious limitations with regard to providing continuously accessible content. This thesis addresses this issue *to improve the download performance and file availability* in such systems.

Chapter 1 "*Introduction*" motivated the benefits of the P2P paradigm with regard to content distribution over the Internet for both content providers and end-users. Alongside this, it also discussed certain prominent limitations, specifically in terms of performance and content availability. Accordingly, the key objective of this thesis was identified and decomposed into a set of research goals.

Chapter 2 "*Background and Related Work*" was structured into three sections. The first section introduced the basic principles and important terminology in the field of P2P content distribution. The second section then gave an abstract overview of a set of prominent P2P protocols used for online content distribution. The information presented formed the basis for understanding the rest of the thesis. Since a core research goal of this thesis was to design a new P2P incentive mechanism, the third section reviewed related work in this domain. To this end, P2P incentive mechanisms were classified into three categories: direct reciprocity (bilateral schemes), indirect reciprocity (multilateral schemes), and virtual currency systems. It was primarily concluded that currently deployed P2P content distribution systems solely rely on bilateral tit-for-tat incentives strategies, because of their ease of implementation and robustness to untruthful peer behaviour.

Chapter 3 "*Measurements and Analysis of the BitTorrent System*" detailed two major measurement studies that we performed in the popular BitTorrent system. The purpose of this was to *(i)* validate our critique of P2P content distribution with regard to providing continuous content distribution and *(ii)* to discover vital properties that cause these limitations. To this end, we found that BitTorrent indeed suffers from large-scale performance and availability problems, affecting millions of users. Specifically, users in BitTorrent download with a median rate of only 30 KBps. Worse, every 12th user is unable to download content at all, confirming very high levels of content unavailability. Our root cause analysis revealed that insufficient incentive design is core reason for the issues observed. In particular, we demonstrated that BitTorrent's tit-for-tat strategy only offers incentives when users are in the process of downloading a file, and can only take place between peers downloading the same file. Thus, users are given no incentive to remain online to serve a file after their download has completed (i.e., to act as seeders). As a consequence, most BitTorrent swarms lose these important users after only a few days. However, as shown by our measurement data, these *seeders* play a vital role in BitTorrent's performance as they provide resources without

consuming any, and ensure that a complete copy of the file remains in the system. Finally, the chapter then concluded that, therefore, any viable solution must incentivise seeders to persist in the system.

Chapter 4 "*Analysis of Solution Space*" studied a set of feasible solutions with regard to the issues found in Chapter 3, both to narrow the solution space and to shape our incentive design in Chapter 5. In this regard, three abstract cross-torrent approaches, as well as single-torrent incentive mechanisms were considered and quantitatively analysed through extensive trace-based simulations. Most notably, it was found that bilateral incentive strategies (i.e., tit-for-tat) are generally insufficient to provide robust incentives for seeding. This is because most users *(i)* do not meet each other repeatedly, and *(ii)* do not simultaneously require each other's content. Instead, it was demonstrated that the only way to overcome performance and availability issues would be to use *multilateral* incentive strategies. These kinds of incentives allow users to contribute data to one user yet receive reciprocation from another. Last, through these insights, a set of key solution requirements were derived.

Chapter 5 "*A New P2P Incentive Mechanism – FairSwarm.KOM*" focussed on the design of a novel P2P incentive mechanism that was devised to meet the solution requirements posed in Chapter 4. Within this chapter, a conceptual overview of FairSwarm.KOM's basic concepts and major building blocks was given. Following this, each building block was presented and discussed in detail, while important design decisions were justified. The result was a generic incentives solution that propagates bidirectional transaction receipts (called coupons) in the network to broaden the users' knowledge base for decision-making.

Chapter 6 "*Evaluation of FairSwarm.KOM*" evaluated the newly devised P2P incentive mechanism. After presenting the aims and methodology, a threefold analysis of FairSwarm.KOM was performed. The first analysis studied workload-dependent parameters controlling performance and overhead aspects of the approach. The second analysis then compared FairSwarm.KOM to BitTorrent's tit-for-tat incentive strategy and an oracle-based incentive mechanism, in which users have global knowledge of the network. Through the use of trace-based simulations, it was shown that, unlike BitTorrent's tit-for-tat incentive, FairSwarm.KOM guarantees almost persistent file availability in all overlay swarms under consideration (i.e., >99% of the users were always able to successfully download content). Furthermore, when compared to BitTorrent's tit-for-tat incentives, the approach yields a median performance improvement of more than 86%. We demonstrated that these two properties are achieved without hurting the fairness of individual users, providing service differentiation proportional to the users' upload contribution. Importantly, this analysis also showed that the oracle-based approach offers very little improvement over FairSwarm.KOM, thereby highlighting the effectiveness of FairSwarm.KOM's coupon distribution. Following this, an overhead analysis was performed to find that FairSwarm.KOM's signalling traffic accounts on average for only 0.11% of a user's total traffic and for 0.63% in the worst case. We also pointed out that this percentage remains constant, irrespective of how much users download or upload, highlighting the scalability of the approach. The final analysis in this chapter studied FairSwarm.KOM's susceptibility to untruthful peer behaviour. Here, it was proven that FairSwarm.KOM provides increased robustness against *(i)* attacks that are common in already existing P2P content distribution systems (i.e., free-riding), and *(ii)* more sophisticated attacks that may arise due to the sharing of information among the peers (e.g., Sybil attacks).

7.2 MAJOR CONTRIBUTIONS

A Novel Incentive Solution to Address the Seeder Promotion Problem

The first major contribution of this thesis is the design and implementation of *FairSwarm.KOM*: a generic P2P incentive solution to address what we call the *seeder promotion problem* [60]. As highlighted in BitTorrent, this problem occurs when users refuse to continue serving content after their own download has been completed (i.e., to act as seeders). Therefore, we have shown that it is the primary reason for the performance and availability issues currently observed in P2P content distribution systems (cf. Chapter 3).

FairSwarm.KOM addresses this problem by building multilateral incentive strategies that are agnostic to time, content, and individual users. However, unlike digital currency systems [75, 113, 123], in which contribution information is globally visible, FairSwarm.KOM uses a one-level of indirection propagation scheme to provide robust incentives for seeding. More precisely, it follows a novel receipt based-approach that persistently documents the bilateral interaction pattern of two nodes in so-called *coupons*. These coupons are then propagated in the network, both to enrich the knowledge base of other users and to allow peers to claim back prior contributions. Instead of simply flooding these coupons in the system, FairSwarm.KOM exploits the long-term interaction patterns of the users and, thus, limits the propagation of coupons to at most one hop in the overlay network. This restriction fosters scalability and provides increased robustness (i.e., to untruthful peer behaviour and peer turnover). Therefore, the devised incentive solution is extremely lightweight and also fully decentralised.

Measurement and Analysis of BitTorrent File Availability and Performance

The next major contribution of this thesis is the measurement-based analysis of the BitTorrent system to ascertain the *causes*, *characteristics*, and *repercussions* of BitTorrent's limitations with regard to providing continuous content distribution. To explore this, two large-scale measurement studies were conducted. The first study investigated BitTorrent on a macroscopic level by periodically probing over 46,000 overlay swarms to *(i)* ascertain their high level characteristics such as swarm size and seeder/leecher ratio, and to *(ii)* obtain important information about the users' arrival patterns. To this end, we collected data on more than 29 million users worldwide. The second study then investigated BitTorrent on a microscopic level by contacting over 700,000 individual peers in 832 torrents to discover relevant properties such as their download rates and chunk availability. These studies go significantly beyond any previous work such as [84, 97, 110, 51, 43, 42, 8] by combining per-node, per-torrent, and system-wide observations. Consequently, this rich measurement data allowed us to extend previous works to obtain far more accurate results (e.g., the causes for file unavailability occurring and the repercussions of this).

Solution Space Analysis of the Seeder Promotion Problem

Another major contribution is the detailed analysis of the role of seeders in BitTorrent. Many recent works have primarily focussed on how to improve the robustness, fairness, and performance properties of BitTorrent's tit-for-tat incentives without effectively offering strong incentives for seeding [72, 77, 78, 112, 93]. This, however, has been shown to be a necessity for guaranteeing high download performance and long-term content availability. The research conducted in this thesis is, therefore, the first that objectively investigates the solution space for dealing with the seeder promotion problem. Specifically, both single-torrent and cross-torrent incentive approaches were investigated to ascertain which is superior based on three key metrics: availability, performance, and fairness. To achieve this, we utilised our BitTorrent measurement

data to execute accurate trace-based simulations for the different abstract solutions considered. Using the results, we ascertained the different trade-offs between the four general solutions: extending seeding times, cross-torrent bartering, local persistent histories, and global shared histories. Finally, with this information, we proposed a set of concrete requirements that a solution must fulfil to overcome the shortcomings observed in BitTorrent. We believe that these requirements can also serve as helpful guidelines for other researchers when attempting to design novel incentives in the area of P2P content distribution.

7.3 OTHER CONTRIBUTIONS

Demonstrating the Feasibility of FairSwarm.KOM

Through the use of trace-based simulations and measurement analysis, we have demonstrated that the design of FairSwarm.KOM is indeed a viable solution to the seeder promotion problem. This means, under current operating environments in BitTorrent, it is guaranteed that FairSwarm.KOM users can recoup their contributions when acting as seeders in torrents that they have previously downloaded from (i.e., cross-torrent seeding). Furthermore, this approach has also been shown to often offer superior performance, availability, and fairness. More generally, this further confirms the feasibility of using third-party observations in P2P systems, as well as the viability of using information gossiping to support this.

Development of a Secure Data Exchange Specification

The success of FairSwarm.KOM crucially depends on the fact that peers obtain coupons for their data provisioning, which can then be disseminated in the system. Consequently, it is of particular importance that the issuing of these transaction receipts is closely tied to the actual delivery of data. To this end, a sophisticated specification has been developed that allows two potentially mistrusting parties to exchange these items in a *fair way*. That is, any possible execution of the specification ends with one of the following two results: *(i)* either the data provider has the corresponding coupon for its service provisioning and, at the same time, the data consumer has received the requested data, or, *(ii)* both peers go away empty-handed. This can be guaranteed even in the presence of communication failures or users attempting to cheat the system.

The Introduction of Pluggable Service Policies

FairSwarm.KOM's design decouples the mechanism prescribing how to use reputation (the Service Policy) from the mechanism providing this information (the Coupon Propagation Specification). This concept allows system designers or individual users to change their own service policy according to any pertinent criteria. In particular, depending on the chosen policy, different objectives can be pursued. For instance, a service policy can be chosen to provide maximum robustness against strategic attacks, optimised for total system throughput or configured to foster better fairness. Also, a mix of all of these aspects is possible, as exemplified by FairSwarm.KOM's proposed default service policy. To aid in this, some brief guidelines have also been presented for system designers when attempting to optimise these criteria (cf. Section 5.6.2).

Proving the Unfeasibility of Single-Torrent Solutions to Address File Unavailability

This thesis has proven that any single-torrent incentive solution that attempts to lengthen users' online times (e.g., with monetary awards or file encryption) will fail to provide long-term availability in BitTorrent. This is because it is only possible to lengthen the user online times by a relatively limited period (e.g., 10 minutes, 1 hour, etc.). However, as shown in this thesis, the user arrival rates in BitTorrent follow

an exponentially decreasing pattern. This means, after a short period of time, the inter-arrival times between two consecutive peers can quickly exceed 24 hours (cf. Section 3.4.3), leaving such single-torrent approaches totally ineffective. In fact, the measurement data indicates that, on average, seeding times would have to be increased by a factor of 10 to gain 99% availability. This requires an average seeding time of more than 34 hours per user.

7.4 RESEARCH GOALS REVISITED

Chapter 1 detailed three core research goals. This section now revisits these to ascertain how effectively they have been fulfilled.

- *To investigate the shortcomings of current P2P content distribution systems with regard to providing continuous content distribution. Specifically, to understand how existing incentive mechanisms fail to fulfil performance and availability requirements.*
  - This research goal has been addressed through the deployment of two major measurement studies in BitTorrent. It was found that existing incentive schemes fail to fulfil performance and availability requirements, because they fail to promote seeding. For that reason, user seeding times are consistently too short to compensate for the exponentially increasing inter-arrival times of users. As a result, overlay swarms inevitably lose seeders over time; a fact that reduces both performance and availability.

- *To design and build a new P2P incentive mechanism that addresses any discovered performance and availability limitations of existing systems.*
  - This research goal has been first addressed by performing a quantitative analysis of the abstract solution space using both trace-based simulations and measurement data analysis. The purpose was to reduce the solution space to gain an understanding of the best general approach to take. Through this, we derived a set of key requirements that a concrete solution must fulfil to overcome the performance and availability issues discovered. The newly designed P2P incentive mechanism FairSwarm.KOM took all of these requirements into account.

- *To show the superiority of the new incentive mechanism; specifically to* (i) *validate that improved performance and availability can be achieved compared to existing tit-for-tat mechanisms, and to* (ii) *ensure that this can take place in a robust, fair, and low overhead manner considering current operating environments.*
  - This research goal has been investigated through the use of up-to-date trace-based simulations, taking our long-term BitTorrent measurement data as an input. More specifically:
    - The FairSwarm.KOM protocol has been accurately implemented in a well-known BitTorrent simulator [10], which has been extended to realistically model a BitTorrent community over a prolonged period of time. The correctness and accurateness of the resulting simulation model has been validated with live experiments.
    - To benchmark FairSwarm.KOM, two extremes in the design space of P2P incentives were considered, *(i)* BitTorrent's tit-for-tat incentive, which operates solely on local observations, and *(ii)* an artificial oracle-based incentive mechanism, through which user have global knowledge of the system.
    - The long-term effectiveness of FairSwarm.KOM with respect to *performance*, *availability*, and *fairness* was shown through the use of simulations based

on our month-long BitTorrent measurement study. It was shown that FairSwarm.KOM performs superior to BitTorrent's tit-for-tat in all three properties and performs close to what the oracle-based system achieves.

* The *light-weightness* of the new incentive mechanism was also demonstrated by evaluating FairSwarm.KOM's signalling traffic and computational overhead. The communication traffic generated is minor compared to the users' service traffic, while the computational overhead of the scheme is negligible.
* The *robustness* of the scheme was demonstrated by considering both *(i)* attacks that are common to already existing P2P content distribution systems and *(ii)* more sophisticated attacks that arise due to use of third-party experiences. All attacks considered were handled effectively through FairSwarm.KOM's default service policy, using one-hop information for decision making.

To conclude, through the fulfilment of these research goals, the key contribution of this thesis has been to strengthen and improve the download performance and file availability of P2P content distribution systems. Importantly, this contribution has been validated through an extensive evaluation, based on up-to-date measurement traces.

7.5 FUTURE WORK

Through the contributions made in this thesis, a very important problem in the area of P2P content distribution has been addressed: offering incentives to ensure the long-term availability and high performance P2P delivery of content. This has been achieved through the development of a novel incentive mechanism, which specifically deals with the seeder promotion problem [60]. However, a number of interesting areas of future work remain; some of which are presented in the following.

The evaluation of FairSwarm.KOM assumes a clean slate approach, in which all users employ the newly designed P2P incentive mechanism using FairSwarm.KOM's default service policy. This policy has been devised to address the research challenges presented in Chapter 1. Concretely, it represents an interesting trade-off between the different design extremes: performance, availability, fairness, and robustness. However, as already stated in Chapter 5, the proposed policy only represents one point in the design space. Since system designers or individual users do not have to follow this recommendation, we have also presented a few variations and included a brief discussion of their pros and cons. Valuable future work could extend this by studying the repercussions of peers mixing these strategies. Furthermore, FairSwarm.KOM's design allows the protocol to be incrementally deployed in already existing P2P content distribution systems. To this end, it would be interesting to see how a single FairSwarm.KOM peer or a small minority of them would perform when tit-for-tat incentive clients hold the majority.

Moreover, in our current implementation of FairSwarm.KOM, file objects have an equal value over time and among users. That is, the amount of data provided is directly stored in the digital receipts (coupons) that are gossiped in the system, irrespective of the time the contribution was made or the object popularity. Another line of future work could be to extend this approach by additionally taking content popularity into account. For instance, a user that acts as a file replica for highly unpopular content could receive more credits than users providing popular content files that are anyway highly replicated in the system.

Despite the above areas of future work, FairSwarm.KOM is a generic incentive solution that can be adapted to a variety of P2P content distribution systems. This thesis, however, has focussed on the integration of FairSwarm.KOM into the BitTorrent protocol. This choice was motivated by the fact that BitTorrent represents the current de-facto standard for scalable content distribution. It is used by millions of users

world-wide and is responsible for transferring terabytes of data per day. Hence, a solution for BitTorrent was seen as the most urgent. An interesting avenue of future research is to extend our work for other popular P2P content distribution systems, such as eMule or Gnutella. This need was also validated, for instance, through the investigation of the eMule file-sharing system, to show that it suffers from serious performance problems [77].

Finally, P2P streaming services have recently gained increasing popularity, as exemplified by P2P IPTV applications such as PPStream[1], PPLive[2], or SOPCast[3]. In this regard, recent work has shown that bilateral incentives do not work effectively for streaming applications [111, 95]. This is because peers usually have different playback positions, meaning that they rarely have a mutual interest in each other, i.e., peers can only download chunks from users that are further in their local playback point, without being able to offer something valuable in return. Therefore, multilateral incentive schemes such as FairSwarm.KOM may offer an effective solution, since uploading data to one user could be rewarded by a different user (who is currently at a later playback point).

7.6 CONCLUDING REMARKS

This thesis has identified and highlighted the need for multilateral incentive schemes in the area of P2P content distribution. In particular, current popular P2P content distribution systems almost exclusively rely on tit-for-tat incentive strategies. As a result, most of these systems suffer from significant performance and availability problems, as exemplified by BitTorrent. This thesis has proposed a novel incentive design that is capable of addressing these serious limitations.

While, in practise, the complexity of multilateral incentive schemes has recently deterred system designers from using such approaches, this thesis has demonstrated that such a sophisticated design can be implemented in an extremely light-weight and robust manner. Moreover, when considering the insights obtained in this thesis, one thing is clear; to fully exploit the potential of P2P systems with regard to providing continuous content distribution, it is often required that users resort to information that goes beyond local histories and direct observation, desirable or not.

1 http://www.ppstream.com
2 http://www.pptv.com
3 http://www.sopcast.com

REFERENCES

[1] Osama Abboud, Aleksandra Kovacevic, Kalman Graffi, Konstantin Pussep, and Ralf Steinmetz. Underlay Awareness in P2P Systems: Techniques and Challenges. In *23rd International Symposium on Parallel and Distributed Processing (IPDPS)*, 2009.

[2] Eytan Adar and Bernardo A. Huberman. Free Riding on Gnutella. *First Monday*, 5:1–18, 2000.

[3] Chris Anderson. The Long Tail: Why the Future of Business is Selling Less of More. Hyperion, 2006.

[4] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36:335–371, 2004.

[5] Demetris Antoniades, Evangelos P. Markatos, and Constantine Dovrolis. One-Click Hosting Services: A File-Sharing Hideout. In *9th Conference on Internet Measurement (IMC)*, 2009.

[6] Arstechnica. iTunes Sells 25% of all Music in the US, 69% of Digital. http://arstechnica.com/apple/news/2009/08/itunes-sells-25-of-all-music-in-the-us-69-of-digital.ars.

[7] Robert Axelrod. The Evolution of Cooperation. Basic Books, 1984.

[8] Anthony Bellissimo, Brian N. Levine, and Prashant Shenoy. Exploring the Use of BitTorrent as the Basis for a Large Trace Repository. Technical Report 04-41, University of Massachusetts, 2004.

[9] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan. Analyzing and Improving BitTorrent Performance. Technical Report MSR-TR-2005-03, Microsoft Research, 2005.

[10] Ashwin R. Bharambe, Cormac Herley, and Venkata N. Padmanabhan. Analyzing and Improving BitTorrent Networks Performance Mechanisms. In *25th International Conference on Computer Communications (INFOCOM)*, 2006.

[11] Ernst W. Biersack, Pablo Rodriguez, and Pascal A. Felber. Performance Analysis of Peer-to-Peer Networks for File Distribution. In *5th International Workshop on Quality of Future Internet Services (QoFIS)*, 2004.

[12] BitComet. BitComet Tracker Overview. http://www.bitcomet.com/tools/tracker/index.htm.

[13] BitTorrent Inc. BitTorrent Strikes Digital Download Deals with 20th Century Fox, G4, Kadokawa, Lionsgate, MTV Networks, Palm Pictures, Paramount and Starz Media. http://www.bittorrent.com/pressreleases/2006/11/28/bittorrent-strikes-digital-download-deals-with-20th-century-fox-g4-kadokawa.

[14] Blizzard. Blizzard Downloader (F.A.Q.). http://www.worldofwarcraft.com/info/faq/blizzarddownloader.html.

[15] Thomas Bocek, Yehia El-khatib, Fabio V. Hecht, David Hausheer, and Burkhard Stiller. Compact PSH: An Efficient Transitive TFT Incentive Scheme for Peer-to-Peer Networks. In *34th Conference on Local Computer Networks (LCN)*, 2009.

[16] Sonja Buchegger and Jean-Yves Le Boudec. A Robust Reputation System for Peer-to-Peer and Mobile Ad-Hoc Networks. In *2nd Workshop on the Economics of Peer-to-Peer Systems (P2PECON)*, 2004.

[17] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *19th Symposium on Operating Systems Principles (SOSP)*, 2003.

[18] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Scott Shenker, and Nick Lanham. Making Gnutella-like Peer-to-Peer Systems Scalable. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2003.

[19] Alice Cheng and Eric Friedman. Sybilproof Reputation Mechanisms. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005.

[20] Clip2. The Gnutella Protocol Specification v0.4 (Document Revision 1.6). http://rfc-gnutella.sourceforge.net/developer/stable/index.html, 2001.

[21] Bram Cohen. Incentives Build Robustness in BitTorrent. In *1st Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, 2003.

[22] Fabrizio Cornelli, Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Choosing Reputable Servents in a Peer-to-Peer Network. In *11th International World Wide Web Conference (WWW)*, 2002.

[23] Vasilios Darlagiannis. *Overlay Network Mechanisms for Peer-to-Peer Systems*. PhD thesis, Technische Universität Darmstadt, 2005.

[24] Vasilios Darlagiannis, Andreas Mauthe, Oliver Heckmann, Nicolas Liebau, and Ralf Steinmetz. On Routing in a Two-Tier Overlay Network based on De-Bruijn Digraphs. In *10th Network Operations and Management Symposium (NOMS)*, 2006.

[25] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing Residential Broadband Networks. In *7th Internet Measurement Conference (IMC)*, 2007.

[26] John Douceur. The Sybil Attack. In *1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[27] Jörg Eberspächer and Rüdiger Schollmeier. First and Second Generation of Peer-to-Peer Systems. In Ralf Steinmetz and Klaus Wehrle, editors, *Peer-to-Peer Systems and Applications*, 2005.

[28] Jörg Eberspächer and Rüdiger Schollmeier. Past and Future. In Ralf Steinmetz and Klaus Wehrle, editors, *Peer-to-Peer Systems and Applications*, 2005.

[29] Shimon Even and Yacov Yacobi. Relations among Public Key Signature Schemes. Technical Report 175, Israel Institute of Technology, 1980.

[30] Jarret Falkner, Michael Piatek, John P. John, Arvind Krishnamurthy, and Thomas Anderson. Profiling a Million User DHT. In *7th Conference on Internet Measurement (IMC)*, 2007.

[31] Bin Fan, Dah-Ming Chiu, and John C. S. Lui. The Delicate Tradeoffs in BitTorrent-like File Sharing Protocol Design. In *International Conference on Network Protocols (ICNP)*, 2006.

[32] Pascal A. Felber and Ernst W. Biersack. Self-scaling Networks for Content Distribution. In *International Workshop on Self-* Properties in Complex Information Systems (SELF-STAR)*, 2004.

[33] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *5th Conference on Electronic Commerce (EC)*, 2004.

[34] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-Riding and Whitewashing in Peer-to-Peer Systems. In *Annual Workshop on Economic and Information Security (WEIS)*, 2004.

[35] Anja Feldmann and Vinay Aggarwal. ISP-Aided Neighbor Selection in Peer-to-Peer Systems. In *IETF Peer-to-Peer Infrastructure Workshop (P2Pi)*, 2008.

[36] Organisation for Economic Cooperation and Development (OECD). Broadband Statistics. http://www.oecd.org/sti/ict/broadband.

[37] Lester Randolph Ford and Delbert Ray Fulkerson, editors. *Flows in Networks*. Princeton University Press, 1962.

[38] Eric J. Friedman and Paul Resnick. The Social Cost of Cheap Pseudonyms. *Journal of Economics and Management Strategy*, 10:173–199, 2000.

[39] Flavio D. Garcia and Jaap-Henk Hoepman. Off-line Karma: A Decentralized Currency for Peer-to-Peer and Grid Applications. In *3rd Applied Cryptography and Network Security Conference (ACNS)*, 2005.

[40] Christos Gkantsidis, John Miller, and Pablo Rodriguez. Anatomy of a P2P Content Distribution System with Network Coding. In *5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[41] Krishna Gummadi, Ramakrishna Gummadi, Steve Gribble, Sylvia Ratnasamy, Scott Shenker, and Ion Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2003.

[42] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. In *5th Internet Measurement Conference (IMC)*, 2005.

[43] Lei Guo, Songqing Chen, Zhen Xiao, Enhua Tan, Xiaoning Ding, and Xiaodong Zhang. A Performance Study of BitTorrent-like Peer-to-Peer Systems. *IEEE Journal on Selected Areas in Communications*, 25:155–169, 2007.

[44] Anwar A. Hamra, Arnaud Legout, and Chadi Barakat. Understanding the Properties of the BitTorrent Overlay. Technical Report 00162088, INRIA, 2007.

[45] Anwar A. Hamra, Nikitas Liogkas, Arnaud Legout, and Chadi Barakat. Swarming Overlay Construction Strategies. In *18th International Conference on Computer Communications and Networks (ICCCN)*, 2009.

[46] David Hausheer, Nicolas Liebau, Andreas Mauthe, Ralf Steinmetz, and Burkhardt Stiller. Token-based Accounting and Distributed Pricing to Introduce Market Mechanisms. In *3rd International Conference on Peer-to-Peer Computing (P2P)*, 2003.

[47] Oliver Heckmann, Ralf Steinmetz, Nicolas Liebau, Alejandro P. Buchmann, Claudia Eckert, Jussi Kangasharju, Max Mühlhäuser, and Andreas Schürr. Qualitätsmerkmale von Peer-to-Peer-Systemen. Technical Report KOM-TR-2006-03, Technische Universität Darmstadt, 2006.

[48] Daniel Hughes, Geoff Coulson, and James Walkerdine. Free-Riding on Gnutella Revisited: The Bell Tolls. *IEEE Distributed Systems Online*, 6:1–13, 2005.

[49] iPoque. Internet Study 2008/09. http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009.

[50] Tomas Isdal, Michael Piatek, Arvind Krishnamurthy, and Thomas Anderson. Leveraging BitTorrent for End Host Measurements. In *8th International Conference on Passive and Active Network Measurement (PAM)*, 2007.

[51] Mikel Izal, Guillaume Uroy-Keller, Ernst W. Biersack, Pascal A. Felber, Anwar A. Hamra, and Luis Garces-Erice. Dissecting BitTorrent: Five Months in Torrent's Lifetime. In *5th International Conference Passive and Active Network Measurement (PAM)*, 2004.

[52] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Hawe. A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems. Technical Report DEC-TR-301, Digital Equipment Corporation, 1984.

[53] JMule. eDonkey Protocol Specification v0.6.2. http://jmule.org/files/eDonkey-protocol-0.6.2.html, 2002.

[54] Seung Jun and Mustaque Ahamad. Incentives in BitTorrent Induce Free Riding. In *3rd Workshop on Economics of Peer-to-Peer Systems (P2PECON)*, 2005.

[55] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust Algorithm for Reputation Management in Peer-to-Peer Networks. In *12th International World Wide Web Conference (WWW)*, 2003.

[56] Sebastian Kaune, Tobias Lauinger, Aleksandra Kovacevic, and Konstantin Pussep. Embracing the Peer Next Door: Proximity in Kademlia. In *8th International Conference on Peer-to-Peer Computing (P2P)*, 2008.

[57] Sebastian Kaune, Jan Stolzenburg, Aleksandra Kovacevic, and Ralf Steinmetz. Understanding BitTorrent's Suitability in Various Applications and Environments. In *1st International Workshop on Computational P2P Networks: Theory and Practice (COMP2P)*, 2008.

[58] Sebastian Kaune, Gareth Tyson, Andreas Mauthe, Nicolas Liebau, and Ralf Steinmetz. Towards a Lightweight Incentive Scheme for Peer-to-Peer Systems. Technical Report KOM-TR-2008-2, Technische Universität Darmstadt, 2008.

[59] Sebastian Kaune, Ruben C. Rumin, Gareth Tyson, Andreas Mauthe, Carmen Guerrero, and Ralf Steinmetz. Unraveling BitTorrent's File Unavailability: Measurements and Analysis. In *10th IEEE International Conference on Peer-to-Peer (P2P)*, 2010.

[60] Sebastian Kaune, Gareth Tyson, Konstantin Pussep, Andreas Mauthe, Aleksandra Kovacevic, and Ralf Steinmetz. The Seeder Promotion Problem: Measurements, Analysis and Solution Space. In *19th International Conference on Computer Communications and Networks (ICCCN)*, 2010.

[61] Tor Klingberg and Raphael Manfredi. Gnutella 0.6 RFC. http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html, 2002.

[62] Dejan Kostic, Adolfo Rodriguez, Jeannie Albrecht, and Amin Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. *SIGOPS Operation Systems Review*, 37:282–297, 2003.

[63] Aleksandra Kovacevic. *Peer-to-Peer Location-based Search: Engineering a Novel Peer-to-Peer Overlay Network*. PhD thesis, Technische Universität Darmstadt, 2009.

[64] Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Globase.KOM - A P2P Overlay for Fully Retrievable Location-based Search. In *7th International Conference on Peer-to-Peer Computing (P2P)*, 2007.

[65] Aleksandra Kovacevic, Oliver Heckmann, Nicolas Liebau, and Ralf Steinmetz. Location Awareness - Improving Distributed Multimedia Communication. *Special Issue of the Proceedings of IEEE on Advances in Distributed Multimedia Communications*, 96:131–142, 2008.

[66] Yoram Kulbak and Danny Bickson. The eMule Protocol Specification. http://www.cs.huji.ac.il/labs/danss/p2p/eMule/emule.pdf, 2005.

[67] Raul Landa. *Incentive-driven QoS in Peer-to-Peer Overlays*. PhD thesis, University College London, 2010.

[68] Raul Landa, David Griffin, Richard G. Clegg, Eleni Mykoniati, and Miguel Rio. A Sybilproof Indirect Reciprocity Mechanism for Peer-to-Peer Networks. In *28th Conference on Computer Communications (INFOCOM)*, 2009.

[69] Seungjoon Lee, Rob Sherwood, and Bobby Bhattacharjee. Cooperative Peer Groups in NICE. *22th International Conference on Computer Communications (INFOCOM)*, 2:1272–1282, 2003.

[70] Arnaud Legout, Guillaume Urvoy-Keller, and Pietro Michiardi. Rarest First and Choke Algorithms Are Enough. In *6th Internet Measurement Conference (IMC)*, 2006.

[71] Arnaud Legout, Nikitas Liogkas, Eddie Kohler, and Lixia Zhang. Clustering and Sharing Incentives in BitTorrent Systems. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2007.

[72] Dave Levin, Katrina LaCurts, Neil Spring, and Bobby Bhattacharjee. BitTorrent is an Auction: Analyzing and Improving BitTorrent's Incentives. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2008.

[73] Jin Li. On Peer-to-Peer (P2P) Content Delivery. *Peer-to-Peer Networking and Applications*, 1:45–63, 2008.

[74] Qiao Lian, Zheng Zhang, Mao Yang, Ben Y. Zhao, Yafei Dai, and Xiaoming Li. An Empirical Study of Collusion Behavior in the Maze P2P File-Sharing System. In *27th International Conference on Distributed Computing Systems (ICDCS)*, 2007.

[75] Nicolas C. Liebau. *Trusted Accounting in Peer-to-Peer Environments - A Novel Token-based Accounting Scheme for Autonomous Distributed Systems*. PhD thesis, Technische Universität Darmstadt, 2008.

[76] Nicolas C. Liebau, Vasilios Darlagiannis, Andreas Mauthe, and Ralf Steinmetz. Token-based Accounting for P2P-Systems. In *Kommunikation in Verteilten Systemen (KiVS)*, 2005.

[77] Nikitas Liogkas, Robert Nelson, Eddie Kohler, and Lixia Zhang. Exploiting BitTorrent for Fun (But Not Profit). In *5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[78] Thomas Locher, Patrick Moor, Stefan Schmid, and Roger Wattenhofer. Free-Riding in BitTorrent is Cheap. In *5th Workshop on Hot Topics in Networks (HotNets)*, 2006.

[79] Thomas Locher, David Mysicka, Stefan Schmid, and Roger Wattenhofer. A Peer Activity Study in eDonkey and Kad. In *International Workshop on Dynamic Networks: Algorithms and Security (DYNAS)*, 2009.

[80] Ivan Martinovic, Christof Leng, Frank A. Zdarsky, Andreas Mauthe, Ralf Steinmetz, and Jens B. Schmitt. Self-Protection in P2P Networks: Choosing the Right Neighbourhood. In *1st International Workshop on Self-Organizing Systems (IWSOS)*, 2006.

[81] Andreas Mauthe and David Hutchison. Peer-to-Peer Computing: Systems, Concepts and Characteristics. *Praxis in der Informationsverarbeitung und Kommunikation (PIK), Special Issue on Peer-to-Peer*, 26:1–10, 2003.

[82] Maxmind. GeoLite Country Database. http://www.maxmind.com/app/geolite-country.

[83] Petar Maymounkov and David Mazieres. Kademlia: A Peer-to-Peer Information System based on the XOR Metric. In *1st International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[84] Daniel S. Menasche, Antonio A. Rocha, Bin Li, Don Towsley, and Arun Venkataramani. Content Availability and Bundling in Swarming Systems. In *5th International Conference on Emerging Networking Experiments and Technologies (CONEXT)*, 2009.

[85] Jacob J. D. Mol, Johan A. Pouwelse, Dick H. Epema, and Henk J. Sips. Free-Riding, Fairness, and Firewalls in Peer-to-Peer File-Sharing. In *8th International Conference on Peer-to-Peer Computing (P2P)*, 2008.

[86] Msnbc. Netflix to Stream Paramount, Lionsgate, MGM movies. http://www.msnbc.msn.com/id/38644039/.

[87] Animesh Nandi, Tsuen-Wan Ngan, Atul Singh, Peter Druschel, and Dan S. Wallach. Scrivener: Providing Incentives in Cooperative Content Distribution Systems. In *6th International Middleware Conference (Middleware)*, 2005.

[88] Mojo Nation. MNet. http://en.wikipedia.org/wiki/Mnet.

[89] National Institute of Standards and Technology. Digital Signature Algorithm. http://www.itl.nist.gov/fipspubs/fip186.htm.

[90] Seth J. Nielson and Scott A. Crosby. A Taxonomy of Rational Attacks. In *4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[91] Chip Online. Vista-Download bringt Microsoft-Server zum Absturz. http://www.chip.de/news/Vista-Download-bringt-Microsoft-Server-zum-Absturz_20299422.html.

[92] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Stanford Digital Library Technologies Project, 1998.

[93] Michael Piatek, Tomas Isdal, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. Do Incentives Build Robustness in BitTorrent. In *4th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[94] Michael Piatek, Tomas Isdal, Arvind Krishnamurthy, and Thomas Anderson. One hop Reputations for Peer to Peer File Sharing Workloads. In *5th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[95] Michael Piatek, Arvind Krishnamurthy, Arun Venkataramani, Richard Yang, David Zhang Alex, and Er Jaffe. Contracts: Practical Contribution Incentives for P2P Live Streaming. In *7th Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[96] Johan A. Pouwelse, Pawel Garbacki, Dick H. Epema, and Henk J. Sips. A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System. Technical Report PDS-2004-007, Delft University of Technology, 2004.

[97] Johan A. Pouwelse, Pawel Garbacki, Dick H. Epema, and Henk J. Sips. The BitTorrent Peer-to-Peer File-Sharing System: Measurements and Analysis. In *4th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2005.

[98] Himabindu Pucha, David G. Andersen, and Michael Kaminsky. Exploiting Similarity for Multi-Source Downloads using File Handprints. In *4th Symposium on Networked Systems Design and Implementation (NSDI)*, 2007.

[99] Konstantin Pussep, Sergey Kuleshov, Christian Groß, and Sergios Soursos. An Incentive-based Approach to Traffic Management for Peer-to-Peer Overlays. In *3rd Workshop on Economic Traffic Management (ETM)*, 2010.

[100] Dongyu Qiu and Ramakrishnan Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2004.

[101] Qwest. Comparison of Qwest High-Speed Internet Plans. http://www.qwest.com/residential/internet/broadbandlanding/compare_plans.html.

[102] Anirudh Ramachandran, Atish das Sarma, and Nick Feamster. An Incentive Compatible Solution for Blocked Downloads in BitTorrent. In *8th Conference on Electronic Commerce (EC)*, 2007.

[103] Rasterbar. Extension Protocol for Bittorrent. http://www.rasterbar.com/products/libtorrent/extension_protocol.html.

[104] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2001.

[105] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21:120–126, 1978.

[106] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *3rd International Conference on Distributed Systems Platforms (Middleware)*, 2001.

[107] Rubén Cuevas Rumín, Michal Kryczka, Ángel Cuevas, Sebastian Kaune, Carmen Guerrero, and Reza Rejaie. Is Content Publishing in Bittorrent Altruistic or Profit-driven? *eprint arXiv:1007.2327*, 2010.

[108] Bruce Schneier. The Blowfish Encryption Algorithm – One Year Later. *Dr. Dobb's Journal*, 1995.

[109] Alex Sherman, Jason Nieh, and Clifford Stein. FairTorrent: Bringing Fairness to Peer-to-Peer Systems. In *5th International Conference on Emerging Networking Experiments and Technologies (CONEXT)*, 2009.

[110] Georgos Siganos, Josep M. Pujol, and Pablo Rodriguez. Monitoring the BitTorrent Monitors: A Bird's Eye View. In *10th International Conference on Passive and Active Network Measurement (PAM)*, 2009.

[111] Thomas Silverston, Olivier Fourmaux, and John Crowcroft. Towards an Incentive Mechanism for P2P Multimedia Live Streaming Systems. In *8th IEEE International Conference on Peer-to-Peer Computing (P2P)*, 2008.

[112] Michael Sirivianos, Jong H. Park, Rex Chen, and Xiaowei Yang. Free-Riding in BitTorrent Networks with the Large View Exploit. In *6th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[113] Michael Sirivianos, Jong H. Park, Xiaowei Yang, and Stanislaw Jarecki. Dandelion: Cooperative Content Distribution with Robust Incentives. In *59th Annual Technical Conference (ATC)*, 2007.

[114] Moritz Steiner, Wolfgang Effelsberg, Taoufik En-Najjary, and Ernst W. Biersack. Load Reduction in the KAD Peer-to-Peer System. In *5th International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2007.

[115] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A Global View of KAD. In *7th Internet Measurement Conference (IMC)*, 2007.

[116] Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*. Springer, 2005.

[117] Ion Stoica, Robert Morris, David Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2001.

[118] Karthik Tamilmani, Vinay Pai, and Alexander E. Mohr. SWIFT: A System with Incentives for Trading. In *2nd Workshop on the Economics of Peer-to-Peer Systems (P2PECON)*, 2004.

[119] The Telegraph. Windows 7 Downloaders Crash Microsoft's Servers. http://www.telegraph.co.uk/technology/microsoft/4209917/Windows-7-downloaders-crash-Microsofts-servers.html.

[120] Wesley W. Terpstra, Jussi Kangasharju, Christof Leng, and Alejandro P. Buchmann. BubbleStorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2007.

[121] Internet End to End Performance (IEPM) Monitoring. Ookla's Speedtest Throughput Measures. https://confluence.slac.stanford.edu/display/IEPM/Ookla's+Speedtest+Throughput+Measures.

[122] Gareth Tyson. *A Middleware Approach to Building Content-Centric Applications*. PhD thesis, Lancaster University, 2009.

[123] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin G. Sirer. KARMA: A Secure Economic Framework for Peer-to-Peer Resource Sharing. In *1st Workshop on the Economics of Peer-to-Peer Systems (P2PECON)*, 2003.

[124] Mea Wang and Baochun Li. How Practical is Network Coding? In *14th IEEE International Workshop on Quality of Service (IWQoS)*, 2006.

[125] Wikipedia. Secure Hash Algorithm (SHA-1). http://de.wikipedia.org/wiki/Secure_Hash_Algorithm.

[126] Haiyong Xie, Richard Yang, Arvind Krishnamurthy, Yanbin Liu, and Abraham Silberschatz. P4P: Provider Portal for Applications. *SIGCOMM Computer Communication Review*, 38:351–362, 2008.

[127] Li Xiong and Ling Liu. PeerTrust: Supporting Reputation-based Trust for Peer-to-Peer Electronic Communities. *IEEE Transactions on Knowledge and Data Engineering*, 16:843–857, 2004.

[128] Beverly Yang and Hector Garcia-Molina. PPay: Micropayments for Peer-to-Peer Systems. In *10th Conference on Computer and Communications Security (CCS)*, 2003.

[129] Xiangying Yang and Gustavo de Veciana. Service Capacity of Peer-to-Peer Networks. In *23rd International Conference on Computer Communications (INFOCOM)*, 2004.

[130] Yan Yang, Alix L. H. Chow, and Leana Golubchik. Multi-Torrent: a Performance Study. In *16th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2008.

[131] Qiao L. Yu, Yu Peng, Mao Yang, Zheng Zhang, Yafei Dai, and Xiaming Li. Robust Incentives via Multi-Level Tit-for-Tat. In *5th International Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[132] Manaf Zghaibeh and Kostas G. Anagnostakis. On the Impact of Peer-to-Peer Incentive Mechanisms on User Behavior. In *8th Conference on Electronic Commerce (EC)*, 2007.

[133] Chao Zhang, Prithula Dhungel, Di Wu, and Keith W. Ross. Unraveling the BitTorrent Ecosystem. *IEEE Transactions on Parallel and Distributed Systems (to appear)*, 2009.

[134] Zhan Zhang, Shigang Chen, and Myungkeun Yoon. MARCH: A Distributed Incentive Scheme for Peer-to-Peer Networks. In *26th International Conference on Computer Communications (INFOCOM)*, 2007.

[135] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.

All web pages cited in this work have been checked in November 2010. However, due to the dynamic nature of the World Wide Web, web pages can change.

APPENDIX

A.1 SESSION TIME ESTIMATION

In this section, we explain the procedure used to calculate the session time of users in our macroscopic measurement data, which is also presented in [107].

As detailed in Section 3.2.3, our multi-tracker crawler connects to the tracker of a given BitTorrent swarm, in order to obtain a random subset of the IP-addresses of the participating users. While this approach is highly efficient to discover new peers in the system, it still imposes some restrictions to compute a user's session time in a given torrent. In fact, if a peer is contained in a tracker report, one can assume that this peer is actually online in the swarm. However, the converse argument does not hold. That is, if the IP-address of given peer is not included in the tracker report, it may still be online. This makes the estimation of the users' session time, however, very inaccurate.

To remedy this, we first define a model to estimate the number of tracker queries (m) required, in order to obtain the IP-address of a user with a given probability $\mathcal{P}$. Specifically, let us assume that we have a torrent with $\mathcal{N}$ peers, and each tracker query returns a random set of $\mathcal{W}$ IP-addresses. Accordingly, if the user is online in the torrent, the probability $\mathcal{P}$ of obtaining its IP-address in m consecutive queries is given by:

$$\mathcal{P} = 1 - \left(1 - \frac{\mathcal{W}}{\mathcal{N}}\right)^{m} \tag{A.1}$$

Furthermore, we find that 90% of the overlay swarms in the Mininova community have at most 165 online nodes. Therefore, we set $\mathcal{N} = 165$, which is an upper bound allowing us to remove the noise introduced by churn. In addition to this, we make a second conservative assumption: in each response, the tracker gives us $\mathcal{W} = 50$ random IPs (in some cases, we obtain up to 200 IP-addresses). With these numbers and the proposed model, we can assure that, if the user is in the torrent, we will discover it in $m = 13$ tracker queries with a probability of more than 99%.

Trace data analysis further reveals that 90% of the tracker queries are less than 18 minutes apart. For that reason, we assume that the time between two consecutive tracker queries is 18 minutes. Moreover, when multiplying this time with the number of queries required (m), we obtain the time span that must pass to consider a peer as offline. Consequently, we consider a given user as offline if its IP-address is not gathered by our crawlers within $\frac{18}{60} \cdot 13 \approx 4$ hours.

To ultimately compute the session of a peer, we assume that it comes online the first time it is reported by the tracker, and stays online until it is not included in the tracker reports for 4 hours.

A.2 INVESTIGATING THE CAUSES OF SWARM RESILIENCE

The bitfield analysis of Section 3.4.2 has identified a notable percentage (14%) of torrents that can maintain availability even without any seeders. This section investigates how they can survive such seedless states (cf. Kaune et al. [59]).

For the purpose of our analysis, we separate torrents into those that survive in the absence of seeders (*resilient torrents*) and those that do not (*susceptible torrents*). We then investigate quantitative properties of these two groups to ascertain how they differ at various points in their lifecycles. All of the identified metrics have been calculated for each group (across all member torrents) every 10 minutes using information from

Metric	Time after torrent's birth			
	6 hours		24 hours	
	Resilient	Susceptible	Resilient	Susceptible
Swarm speed (in KBps)	95.72	53.50	62.99	44.82
Seeder/Leecher ratio	0.19	0.19	0.32	0.43
Firewalled/NATed peers (in %)	51.30	56.05	54.94	58.44
Distribution Entropy $E(T)$	0.91	0.90	0.92	0.91
Least replicated chunk (# of copies)	15.34	14.21	21.33	23.55
Churn factor CF	0.07	0.11	0.08	0.07
Online leechers	134.05	82.71	163.12	89.55
Online seeders	12.05	11.25	18.17	21.34

Metric	Time before seedless state			
	1 hour		6 hours	
	Resilient	Susceptible	Resilient	Susceptible
Swarm speed (in KBps)	58.83	23.88	68.58	24.99
Seeder/Leecher ratio	0.15	0.03	0.14	0.04
Firewalled/NATed peers (in %)	70.86	61.09	62.30	60.67
Distribution Entropy $E(T)$	0.93	0.94	0.92	0.93
Least replicated chunk (# of copies)	9.21	1.61	8.38	2.82
Churn factor CF	0.03	0.21	0.08	0.15
Online leechers	281.34	111.61	250.48	101.55
Online seeders	5.28	1.51	6.11	1.80

Table 21: Characteristics of resilient torrents (those that maintain availability in seedless state) and susceptible torrents (those that cannot reconstruct the file).

the microscopic traces and the tracker reports. These values have then been averaged together over each time period investigated.

Table 21 gives an overview of all metrics used in this analysis. We calculate these over two time periods: the beginning of the torrents' lifecycle and just before the last seeder goes offline. Although not included in the table, we also investigated the effects of file size and content type without ascertaining any correlation. Most metrics are straightforward, however, two require some explanation: Distribution Entropy ($E(T)$) and the Churn Factor (CF).

The $E(T)$ investigates the distribution of chunks within the swarm; this is to investigate whether torrents that can survive achieve a superior distribution of chunks. We therefore characterize the distribution entropy in a torrent T at a given time t by introducing the following *Entropy Index*:

$$E(T) = \frac{\left(\sum_{j=1}^{P}\sum_{i=1}^{N} V_{ij}\right)^2}{P \cdot \sum_{j=1}^{P}\left(\sum_{i=1}^{N} V_{ij}\right)^2} \tag{A.2}$$

Recall that N defines the number of nodes in the swarm, P is the number of chunks a file is composed of and V_i is the bitfield of node i. This index is similar to Jain's Fairness Index [52] and achieves a value of 1 if all chunks are equally distributed among the peers.

The Churn Factor CF investigates whether torrents that can survive have more stable populations. This factor is defined by N_{disc}/N_{all} where N_{disc} is the number of users that have left the swarm during a given time period (t) and N_{all} is the total number of users observed during this same period. A factor of 0 indicates that no user disconnected within t; by default $t = 10\ mins$.

From the data in Table 21, we can make the following important observations,

- *Torrent Popularity:* From the beginning, resilient torrents exhibit higher leecher population sizes. Larger torrents possess an increased probability of replicating rare chunks before the loss of seeders.
- *Low Churn Factor:* High churn in small torrents creates a greater risk of losing vital chunks; if this coincides with the loss of a seeder then it becomes impossible to recover these chunks again until a seeder returns. Resilient torrents have significantly lower churn factors than susceptible torrents.
- *Seeder/Leecher Ratio:* Resilient torrents exhibit a higher seeder/leecher ratio and, as a derivative of this, experience download rates that are over twice as high as susceptible torrents. This superior performance is highly beneficial for the survival of chunk replicas as it allows the quick duplication of rare chunks. Before seedless state occurring, resilient torrents therefore have many more replicas of the rarest chunk when compared to susceptible torrents.

In summary, these results show that swarm resilience is a product of large, stable populations that can achieve higher download rates due to beneficial seeder/leecher ratios. The combination of these factors results in rarest chunk replication rates that are over 5 times greater than their susceptible counterparts. This makes such swarms highly resilient to the loss of any seeders. Importantly, it also can be concluded that unavailability cannot be addressed by modifying any of BitTorrent's algorithms (e.g., chunk selection) but, instead, must be solved by incentivising users to modify their behaviour. This is exemplified by the lack of any correlation between resilience and distribution entropy.

A.3 SECURITY ANALYSIS OF DATA EXCHANGE SPECIFICATION

In this section, we prove different security properties of FairSwarm.KOM's data exchange specification. In order to understand these proofs, it is recommended to first read the protocol specifications presented in Section 5.5.

Lemma A.3.1 *For a dishonest consumer* C*, it is impossible to obtain data object* O*, without sending the correctly prepared evidence of receipt.*

Proof From Section 5.5.4, it is easy to see that, when the consumer denies to send the correctly prepared evidence of receipt $Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$ in message flow (e3), the provider will immediately run the abort protocol to cancel the transaction. If this happens, the consumer loses its last opportunity to get the decryption key K, both from the provider and the mediator. Nevertheless, there is still some time in between that the consumer could use to illegally obtain object O.

First, consumer C could try to decrypt the received ciphertext O_K using brute force attacks. However, it is assumed that provider P uses a secure symmetric encryption scheme for the encryption of O (cf. Section 5.5.3). Thus, it is impossible for consumer C to defeat this encryption. Consequently, the only way to decrypt O_K is to obtain the encryption/decryption key K. In general, key K can be obtained by `(i)` defeating the asymmetric encryption scheme that is used to encrypt K_M, `(ii)` through provider P, or `(iii)` through mediator M.

`Case (i)`: It is assumed that K_M is protected by the use of secure asymmetric cryptographic algorithms, as detailed in Section 5.5.3. Thus, it is impossible to defeat this encryption.

`Case (ii)`: To obtain the session key K from provider P, node C must sent a *correctly* prepared evidence of receipt in message flow (e3). However, through the use of hashed identities (cf. Section 5.3.2), this digital evidence can be only properly prepared if C gives its signature on the newly generated coupon $\zeta_{P,C}$ using its correct identity. That is, any fraud attempt by using the identity of another node (e.g., peer C′) to obtain key K will be detected by provider P. Therefore, it is impossible for node C to illegally obtain key K from the provider P.

`Case (iii)`: To obtain key K from the mediator, dishonest node C may directly run the recovery protocol upon obtaining ciphertext O_K in message flow (e2). To this end, node C may proceed as follows:

1. it properly sends message flow (r1), including the correctly prepared evidence of receipt $Sig_C\left(\zeta_{P,C}, Sig_P(\zeta_{P,C})\right)$ (cf. Table 11).
2. it properly sends message flow (r1), but prepares the evidence of receipt incorrectly.
3. it may attempt to manipulate message flow (r1) such that the mediator reveals the desired key K mistakenly.

In situation (1), node C indeed obtains key K, but provider P also receives the evidence of receipt (cf. Table 11). Thus, the fraud attempt fails.

In situation (2), the signature checks performed by mediator M will simply fail. As a consequence, the request of node C will be ignored and C receives nothing (cf. Table 11).

In situation (3), node C may attempt to manipulate the data of message flow (r1) to illegally obtain key K. However, since node C does not possess key K, it cannot adapt or newly prepare K_M that is sent in message flow (r1). Thus, to obtain key K, it must forward K_M as received from provider P in message flow (e1). Because of this fact, however, it is also impossible for node C to manipulate the remaining data objects of flow (r1). This is because upon receiving flow (r1), mediator M will use this data to compute the unique transaction label L′. Subsequently, it compares L′ with $K_M = (L, K)$. Accordingly, if C has manipulated only one data object of flow (r1), it is $L \neq L'$ and mediator M will simply ignore C's request. For that reason, it is also impossible for consumer C to illegally obtain key K from mediator M.

To conclude, the analysis of case `(i)`, `(ii)` and `(iii)` have shown that it is impossible for a dishonest consumer C to illegally obtain data object O, without sending the correctly prepared evidence of receipt. ∎

Lemma A.3.2 *For a dishonest provider* P*, it is impossible to obtain a valid coupon* $\zeta_{P,C}$*, without sending the correct encryption key* K*.*

Proof As depicted in Table 9, once consumer C `(i)` receives an invalid secret key K via message flow (e4), `(ii)` does not receive message flow (e4) timely, or, alternatively, `(iii)` the data encryption with the received key K fails (i.e., due to the sending of a wrong key k), node C will contact the mediator for help. In either case, the mediator then checks the honesty of provider P.

In particular, in case of situation `(i)` and `(ii)`, the consumer is supposed to run the recovery protocol, in order to obtain the secret key K with the aid of the mediator. To this end, the mediator tests whether K_M is properly prepared by provider P (cf. Section 5.5.6). If invalid, the mediator issues receipt $REV_{\zeta_{P,C}}$ that revokes C's signature on the most recent version of coupon $\zeta_{P,C}$. Thus, the consumer's signature on coupon $\zeta_{P,C}$, that is (illegally) received in message flow (e3), becomes useless for provider P.

In case of situation (iii), the consumer is supposed to run the complaint protocol by sending ciphertext O_K to the mediator (cf. Section 5.5.7). The mediator then decrypts O_K: $O' = d_K(O_K)$ to subsequently test whether $h(O) = h(O)'$. If this check fails, it is proven that provider P has cheated in the system and the mediator issues the revocation receipt $REV_{\zeta_{P,C}}$. Similarly to the recovery protocol; the illegally obtained signature for coupon $\zeta_{P,C}$ becomes then useless for provider P.

From the above, it is therefore intuitive to see that it is impossible for provider P to obtain a *valid* coupon $\zeta_{P,C}$, without sending the correct encryption key K. ∎

Lemma A.3.3 *For a dishonest provider* P*, it is impossible to upload garbage data, and yet obtain a valid coupon* $\zeta_{P,C}$ *for this illegal service provisioning.*

Proof As detailed in Section 5.5.4, before peer P and C exchange a data object O, a unique transaction label L is generated to which provider P must commit itself (i.e., it signs (L, K_M)). Label L contains the hash value of the updated coupon, as well as the hash of the data object O to be transferred. Consumer C gives its signature on the updated version of coupon $\zeta_{P,C}$ if and only if (i) L is correctly prepared and (ii) provider P has provided its valid signature on this digital commitment. Accordingly, when provider P uploads encrypted garbage data O'_K in message flow (e2), consumer C will detect this fraud attempt upon receiving session key K in flow (e4). This is because the hash value of the obtained data object O' will not match with the value of the data object O that peer C has requested ($h(O') \neq h(O)$). As a consequence, peer C runs the complaint protocol by sending K_M, O'_K, as well as the provider's signature on K_M in message flow (c1) to the mediator (cf. Table 12). The mediator then computes the transaction label L and decrypts O'_K to verify consumer C's complaint. If $h(O')$ does indeed not match to the hash value contained in label L, it is proven that provider P has cheated consumer C. Subsequently, receipt $REV_{\zeta_{P,C}}$ is sent to node C, making coupon $\zeta_{P,C}$ useless for provider P.

Finally, as a last alternative, provider P could also run the abort protocol, upon sending garbage data in flow (e2) and receiving C's signature on coupon $\zeta_{P,C}$ in flow (e3) (cf. Table 9). When running the abort protocol, however, the mediator will send receipt $REV_{\zeta_{P,C}}$ to consumer C (cf. Table 10). This revokes C's signature on coupon $\zeta_{P,C}$, making coupon $\zeta_{P,C}$ again useless for provider P.

Thus, in any case, it is impossible for provider P to upload garbage, and yet obtain a valid coupon $\zeta_{P,C}$ for this illegal service provisioning. ∎

A.4 ANALYSING USER INTERACTION PATTERN

Section 5.4.1 has detailed that the users' knowledge about the network explosively increases with each level of indirection. On the other hand, it has also been discussed that this comes at the expense of vulnerability to attacks and scalability issues. To find a good 'deal' between these trade-offs, an important research question is: *What level of indirection is required so that users can actually claim back previous contributions?* For instance, if most of the peers are unable to recognise previous contributions of other nodes, the chosen level of indirection is clearly too small. Instead, it must by all available means be increased, even though this might impair robustness and scalability.

In one-hop indirect reciprocation, peers can only claim back contributions when situated in a triangular relationship to each other, e.g., $A \rightarrow I \rightarrow B$. Thus, they must share a common interaction partner, currently or in the past. Two-hop indirection, on the other hand, is less restrictive and increases the probability for reciprocation. This is because it also detects cycles with two-hop intermediaries in the contribution graph (e.g., $A \rightarrow I_1 \rightarrow I_2 \rightarrow B$), in addition to the one-hop relationships. The same applies for any higher level of indirection. It can therefore be deduced that the answer to the above question is clearly a matter of the interaction pattern of the nodes. For instance,

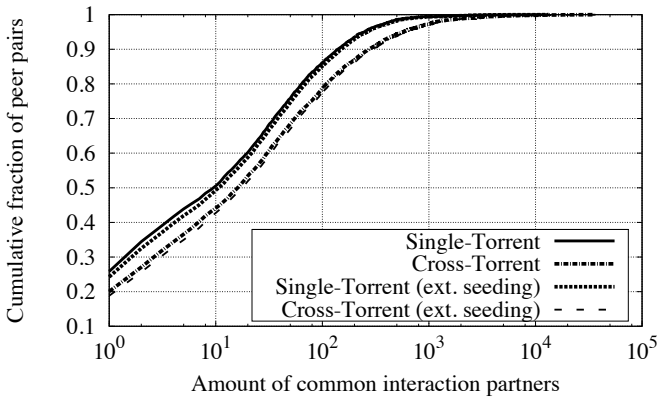


Figure 56: Amount of common prior interaction partners in the BitTorrent system.

Download behaviour	Seeding times	
	As measured	Extended (by one hour)
Active in original torrents	Single-Torrent	Single-Torrent (ext. seeding)
Active in original and former torrents	Cross-Torrent	Cross-Torrent (ext. seeding)

Table 22: Overview of different behavioural strategies.

if the interaction graph of the peers is uniform random, several levels of indirections may be needed to find any cyclic contribution cycle.

Through extensive trace analysis of our one month long BitTorrent trace data, we find, however, that the interaction pattern of current P2P content distribution systems exhibit small world patterns. Specifically, it can be observed that the interaction graph has a clearly clustered structure. This occurs because 79.21% of users fetch content always from the same content category (e.g., Movies, eBooks, Pictures, etc.). Thus, the observed interaction clusters mainly arise due to the users' social interests. Second, it can also be noted that these clusters are interconnected through a number of 'power' users that participate in a wealth of BitTorrent swarms, originating from different content categories. For instance, the top 10% of multi-torrent users download more than 32 torrents on average and, thus, account for 52.23% of the total system demand. Accordingly, these users interact with up to several thousand users in only one month.

These findings clearly suggest that many users are closely adjacent in the interaction graph. To affirm this hypothesis, we performed a trace data analysis to test the effectiveness of indirect reciprocation, particularly with one level of indirection. We were interested to find out whether users share *common* interaction partners in the past, when newly encountering each other. Due to the memory and computational complexity of this analysis, we chose random peer pairs from the measurement data, instead of performing this analysis for all nodes in our trace logs.

Figure 56 shows the cumulative distribution of the amount of common interaction partners, as obtained from all examined peer pairs. To further investigate the possible influence of novel incentive mechanisms (e.g., incentives for seeders), the figure also takes two variations of peer behaviour into account: extended seeding times and cross-torrent collaboration. These two variations lead to four possible combinations as depicted in Table 22.

As can be seen, even in the case of standard BitTorrent (combination: 'Single-Torrent'), which corresponds to the measurements with direct reciprocity, from 91,239 randomly chosen peer pairs only 23,538 (25.8%) did not have a common interaction partner. Hence, about 74% of all peer pairs share an one-hop relationship, that could be exploited

to significantly improve a peer's standing in the system. Also, 96.3% of these users encounter such one-hop interaction partners already in the next file download. Only 3.7% have to wait for two file downloads and more. Thus, the majority of peers could experience reciprocation for previous contributions very quickly. Finally, assuming prolonged online times and cross-torrent collaboration (combination: 'Cross-Torrent ext. seeding'), even up to 81% of the peer pairs are connected indirectly over one hop.

To conclude, due to the small-world like interaction pattern of the nodes, one level of indirection already suffices to enable indirect reciprocation for most of the users.

A.5 ACCURACY OF THE SIMULATION MODEL OF BITTORRENT

In order to validate the correctness of the simulation results produced by the OctoSim BitTorrent simulator of Microsoft Research [9], we have re-simulated an overlay swarm from a well-known BitTorrent measurement study of Legout et al. [71]. In this work, the authors perform a wealth of experiments in the PlanetLab testbed to explore the characteristics of important mechanisms associated to BitTorrent. These experiments include a study of BitTorrent's effectiveness to find peers with similar bandwidth capacities, the interdependency between upload contribution and download reward in BitTorrent, and the importance of the revised unchoking mechanism in seeder state.

According to the work of Legout et al. [71], we consider an overlay swarm consisting of a single initial seeder and 40 leechers. These users are classified into three groups: slow, medium, and fast. In particular, 13 'slow' leechers have an upload capacity of 20 KBps, 14 'medium' leechers have 50 KBps, while the remaining 14 'fast' leechers (including the initial seeder) are endowed with 200 KBps upload capacity. Also, analogous to the work of Legout et al. [71], we provision leechers with infinite download capacity. Furthermore, Legout et al. noted that, in their experiments, all peers joined the overlay swarm at the same time. Therefore, in our experiments, we bring up peers within one second. This is probably closer to what happened in Legout et al.'s experiments, considering the TCP connection setup time, as well as network latencies. Finally, as stated in [71], the file to be downloaded by the peers is 113 Megabytes in size divided into 453 chunks, 256 Kilobytes each.

Legout et al. have provided several result plots in their paper. Six of these have been re-produced with the OctoSim simulator (cf. Figures 57 - 62). All of these plots (with the exception of Figure 62) are averaged results of ten simulation runs using different random seeds. As mentioned previously, in their measurement study, Legout et al. were especially interested in studying the clustering behaviour of peers in BitTorrent. Different aspects of this behaviour are demonstrated in Figure 57, 58, and 59. Specifically, Figure 57 shows the clustering index, which is defined for a given peer in a given class (fast, medium, or slow) as the 'ratio of the duration of regular unchokes to the peers of its class over the duration of regular unchokes to all peers' [71]. Accordingly, a high clustering index indicates a strong preference to upload to peers in the same class.

In general, the simulation results show that all peers favour users from their own bandwidth class for uploading (cf. Figure 57b). We can also notice that the slow peers slightly prefer the medium over the fast ones, while the fast peers slightly prefer the medium over the slow ones. In contrast, the medium peers do not prefer any of the two other groups. All of these facts are also visible in Legout et al.'s version of the plot (cf. Figure 57a).

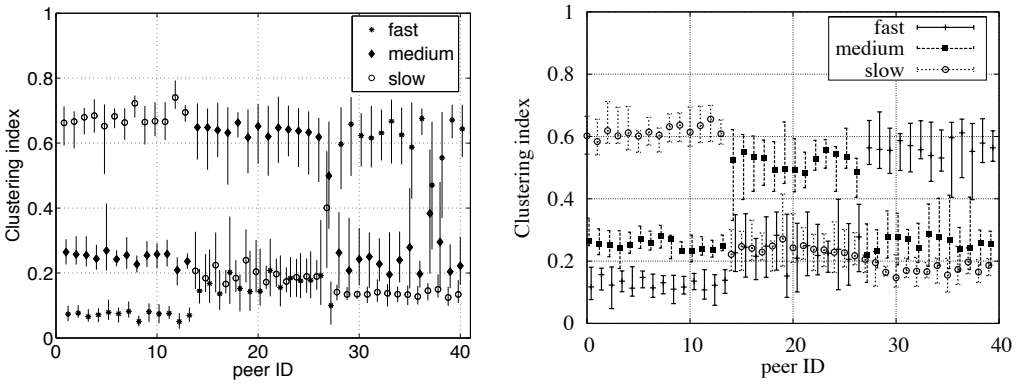
Furthermore, for each individual peer, Figure 58 plots the total time peers unchoked each other via a regular unchoke. In addition to this, Figure 59 shows the amount of data peers have uploaded to each other. As in Legout et al.'s experiments (cf. Figure 58a and 59a), Figure 58b and 59b show a pattern of three darker squares along the diagonal. This suggests that peers cluster inside their bandwidth class,

highlighting that the simulator very realistically reflects the clustering behaviour of peers in BitTorrent.

In addition to this, Figure 60 shows the downloading speed for each individual peer. Because of the clustering pattern, the users' download speed is clearly determined by their uploading bandwidth. To this end, Figure 61 also shows the cumulative distribution function of the download completion times for each bandwidth class. In general, we observe that the simulated times are comparable to those reported by Legout et al., although the simulator tends to produce slightly shorter completion times. This is, however, expected as the simulator does not model network latencies and BitTorrent signalling traffic [9].

Finally, Figure 62 shows the duration of unchokes performed by the initial seeder. It can be observed that the simulated and measured results are almost identical; leechers are unchoked in a uniform manner, irrespective of their upload speed. This is expected, as the 'new' seeder state choking algorithm, described by Legout et al. [71], is used in our experiments.

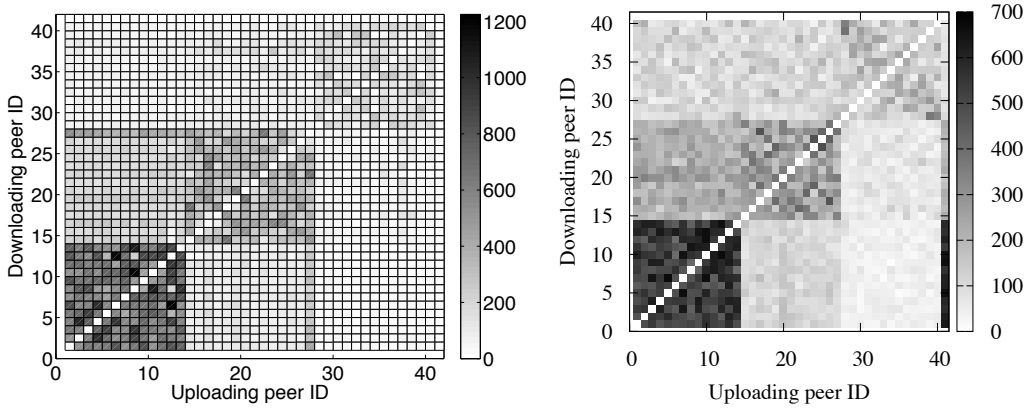
In summary, the simulation results clearly demonstrate that the OctoSim simulator very realistically reflects the BitTorrent protocol. Specifically, all results produced are close to what Legout et al. have measured in their live experiments.



(a) Measurement results of Legout et al. [71].

(b) Simulation results.

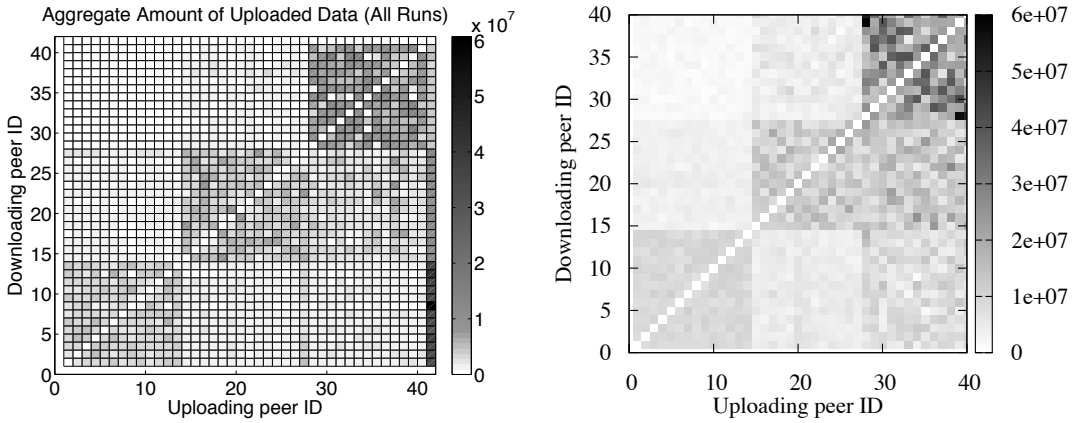
Figure 57: Clustering index for all peers. Note that Subfigure (b) is a reproduction of Figure 2 of Legout et al. [71] (cf. Subfigure (a)). The data is averaged over ten runs with different random seeds. The error bars represent the 10th and 90th percentiles. Peers 1 to 13 represent the 'slow' peers (20 KBps upload capacity), peers 14 to 27 the 'medium' peers (50 KBps upload capacity), and peers 28 to 40 the 'fast' peers (200 KBps upload capacity).



(a) Measurement results of Legout et al. [71].

(b) Simulation results.

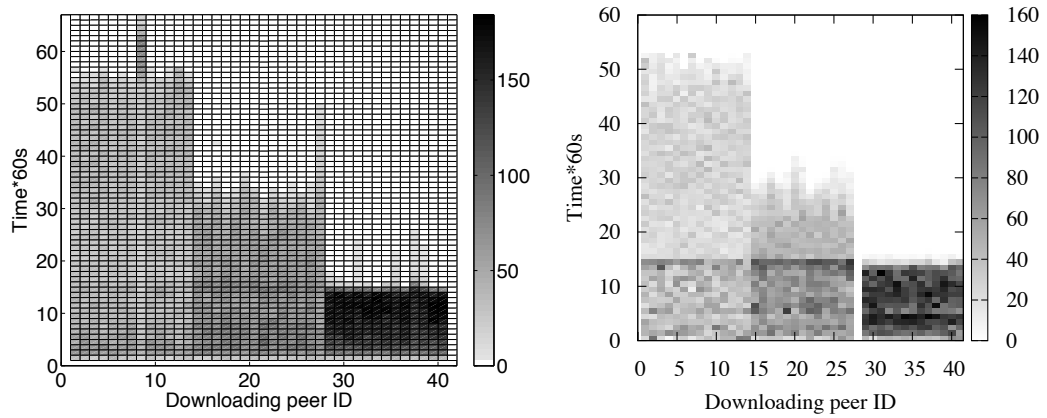
Figure 58: Time duration that peers unchoked each other via a regular unchoke. Note that Subfigure (b) is a reproduction of Figure 1 of Legout et al. [71] (cf. Subfigure (a)). The data is averaged over ten runs with different random seeds. Peers 1 to 13 represent the 'slow' peers (20 KBps upload capacity), peers 14 to 27 the 'medium' peers (50 KBps upload capacity), and peers 28 to 40 the 'fast' peers (200 KBps upload capacity).



(a) Measurement results of Legout et al. [71].

(b) Simulation results.

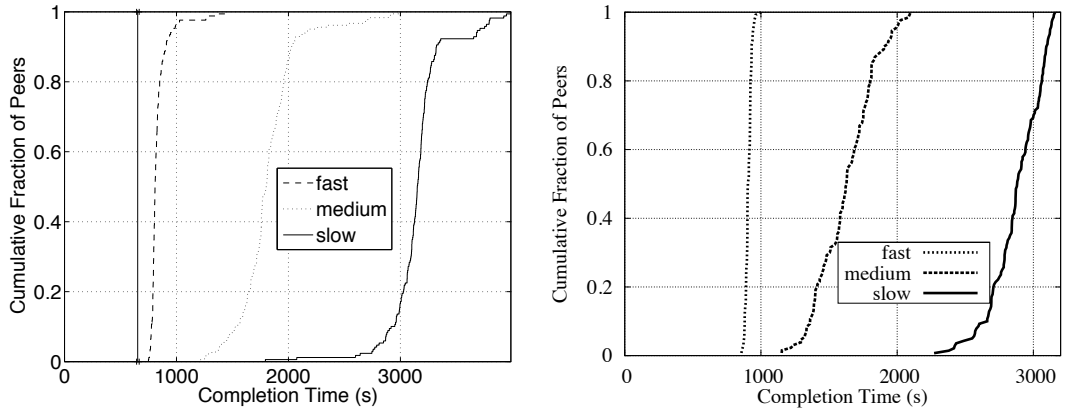
Figure 59: Total amount of data (in bytes) peers uploaded to each other. Note that Subfigure (b) is a reproduction of Figure 5 of Legout et al. [71] (cf. Subfigure (a)). The data is averaged over ten runs with different random seeds. Peers 1 to 13 represent the 'slow' peers (20 KBps upload capacity), peers 14 to 27 the 'medium' peers (50 KBps upload capacity), and peers 28 to 40 the 'fast' peers (200 KBps upload capacity).



(a) Measurement results of Legout et al. [71].

(b) Simulation results.

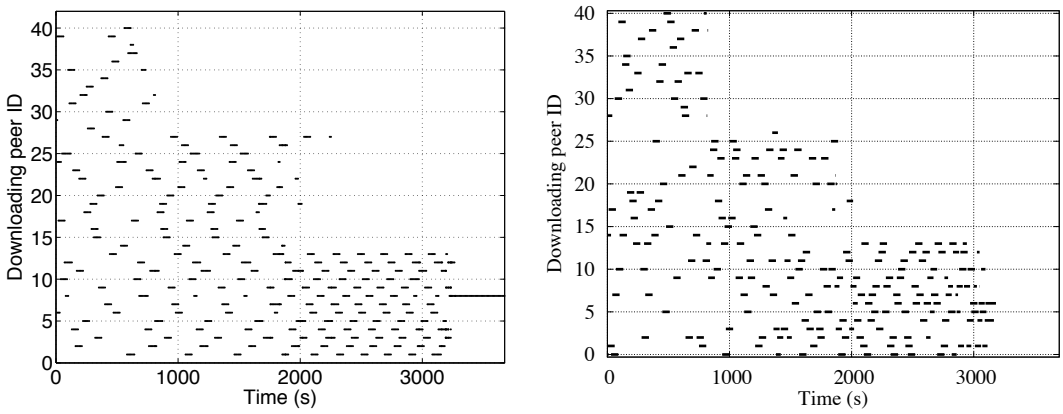
Figure 60: Downloading speed of the users over time (gray scaling is KBps). Note that Subfigure (b) is a reproduction of Figure 4 of Legout et al. [71] (cf. Subfigure (a)). The data is averaged over ten runs with different random seeds. Peers 1 to 13 represent the 'slow' peers (20 KBps upload capacity), peers 14 to 27 the 'medium' peers (50 KBps upload capacity), and peers 28 to 40 the 'fast' peers (200 KBps upload capacity).



(a) Measurement results of Legout et al. [71].

(b) Simulation results.

Figure 61: The cumulative distribution function of peer download times. Note that Subfigure (b) is a reproduction of Figure 3 of Legout et al. [71] (cf. Subfigure (a)). The data is averaged over ten runs with different random seeds.



(a) Measurement results of Legout et al. [71].

(b) Simulation results.

Figure 62: Seeder unchoke duration to each peer, as obtained from a representative experiment run. Note that Subfigure (b) is a reproduction of Figure 7 of Legout et al. [71] (cf. Subfigure (a)). Peers 1 to 13 represent the 'slow' peers (20 KBps upload capacity), peers 14 to 27 the 'medium' peers (50 KBps upload capacity), and peers 28 to 40 the 'fast' peers (200 KBps upload capacity).

B AUTHOR'S PUBLICATIONS

B.1 MAIN PUBLICATIONS

[1] Sebastian Kaune, Ruben Cuevas Rumin, Gareth Tyson, Andreas Mauthe, Carmen Guerrero, and Ralf Steinmetz. Unraveling BitTorrent's File Unavailability: Measurements and Analysis. In *10th International Conference on Peer-to-Peer Computing (P2P)*. 2010.

[2] Sebastian Kaune, Gareth Tyson, Konstantin Pussep, Andreas Mauthe, Aleksandra Kovacevic, and Ralf Steinmetz. The Seeder Promotion Problem: Measurements, Analysis and Solution Space. In *19th International Conference on Computer Communications and Networks (ICCCN)*. 2010.

[3] Sebastian Kaune, Matthias Wählisch, and Konstantin Pussep. Modeling the Internet Delay Space and its Application in Large Scale P2P Simulations. In Klaus Wehrle, Mesut Günes, and James Groß, editors, *Modeling and Tools for Network Simulation*. 2010.

[4] Sebastian Kaune, Konstantin Pussep, Christof Leng, Aleksandra Kovacevic, Gareth Tyson, and Ralf Steinmetz. Modelling the Internet Delay Space Based on Geographical Locations. In *17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. 2009.

[5] Sebastian Kaune, Tobias Lauinger, Aleksandra Kovacevic, and Konstantin Pussep. Embracing the Peer Next Door: Proximity in Kademlia. In *8th International Conference on Peer-to-Peer Computing (P2P)*. 2008.

[6] Sebastian Kaune, Konstantin Pussep, Gareth Tyson, Andreas Mauthe, and Ralf Steinmetz. Cooperation in P2P Systems through Sociological Incentive Patterns. In *Third International Workshop on Self-Organizing Systems (IWSOS)*. 2008.

[7] Sebastian Kaune, Jan Stolzenburg, Aleksandra Kovacevic, and Ralf Steinmetz. Understanding BitTorrent's Suitability in Various Applications and Environments (best paper awarded). In *1st International Workshop on Computational P2P Networks: Theory and Practice (COMP2P)*. 2008.

[8] Sebastian Kaune, Gareth Tyson, Andreas Mauthe, Nicolas Liebau, and Ralf Steinmetz. Towards a Lightweight Incentive Scheme for Peer-to-Peer Systems. Technical Report KOM-TR-2008-2, Technische Universität Darmstadt, 2008.

B.2 OTHER PUBLICATIONS

[1] Konstantin Pussep, Sebastian Kaune, Osama Abboud, Christian Huff, and Ralf Steinmetz. On Energy-Awareness for Peer-assisted Streaming with Set-Top Boxes. In *6th International Conference on Network and Service Management (CNSM)*. 2010.

[2] Konstantin Pussep, Christof Leng, and Sebastian Kaune. Modeling User Behavior in P2P Systems. In Klaus Wehrle, Mesut Günes, and James Groß, editors, *Modeling and Tools for Network Simulation*. 2010.

[3] Ruben Cuevas Rumin, Michal Kryczka, Angel Cuevas Rumin, Sebastian Kaune, Carmen Guerrero, and Reza Rejaie. Is Content Publishing in BitTorrent Altruistic or Profit Driven? In *6th International Conference on Emerging Networking Experiments and Technologies (CONEXT)*. 2010.

[4] Konstantin Pussep, Sebastian Kaune, Jonas Flick, and Ralf Steinmetz. A Peer-to-Peer Recommender System with Privacy Constraints. In *3rd International Workshop on P2P, Parallel, Grid and Internet Computing (3PGIC)*. 2009.

[5] Gareth Tyson, Andreas Mauthe, Sebastian Kaune, Mu Mu, and Thomas Plagemann. Corelli: A Peer-to-Peer Dynamic Replication Service for Supporting Latency-Dependent Content in Community Networks. In *16th Annual Multimedia Computing and Networking (MMCN)*. 2009.

[6] Kalman Graffi, Sebastian Kaune, Konstantin Pussep, Aleksandra Kovacevic, and Ralf Steinmetz. Load Balancing for Multimedia Streaming in Heterogeneous Peer-to-Peer Systems. In *18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*. 2008.

[7] Aleksandra Kovacevic, Kalman Graffi, Sebastian Kaune, Christof Leng, and Ralf Steinmetz. Towards Benchmarking of Structured Peer-to-Peer Overlays for Network Virtual Environments. In *14th International Conference on Parallel and Distributed Systems (PDP)*. 2008.

[8] Konstantin Pussep, Sebastian Kaune, Christof Leng, Aleksandra Kovacevic, and Ralf Steinmetz. Impact of User Behavior Modeling on Evaluation of Peer-to-Peer Systems. Technical Report KOM-TR-2008-07, Technische Universität Darmstadt, 2008.

[9] Gareth Tyson, Paul Grace, Andreas Mauthe, and Sebastian Kaune. The Survival of the Fittest: An Evolutionary Approach to Deploying Adaptive Functionality in Peer-to-Peer Systems. In *7th Workshop on Adaptive and Reflective Middleware (ARM)*. 2008.

[10] Kalman Graffi, Konstantin Pussep, Sebastian Kaune, Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Overlay Bandwidth Management: Scheduling and Active Queue Management of Overlay Flows. In *32th Conference on Local Computer Networks (LCN)*. 2007.

[11] Aleksandra Kovacevic, Sebastian Kaune, Patrick Mukherjee, Nicolas Liebau, and Ralf Steinmetz. Benchmarking Platform for Peer-to-Peer Systems. *IT - Information Technology: Methods and Applications of Informatics and Information Technology*, 49:312–319, 2007.

[12] Nicolas Liebau, Konstantin Pussep, Kalman Graffi, Sebastian Kaune, Eric Jahn, André Beyer, and Ralf Steinmetz. The Impact of the P2P Paradigm. In *13th Americas Conference on Information Systems (AMCIS)*. 2007.

C

CURRICULUM VITAE

PERSONAL

Name	Sebastian Kaune
Date of Birth	November 14th, 1979
Place of Birth	Darmstadt, Germany
Nationality	German

EDUCATION

since 01/2007	Technische Universität Darmstadt, Germany Doctoral candidate at the Department of Electrical Engineering and Information Technology
10/2000–12/2006	Technische Universität Darmstadt, Germany Studies of Computer Science
9/1990–6/1999	Viktoriaschule, Darmstadt, Germany Degree: Allgemeine Hochschulreife

WORK EXPERIENCE

Since 4/2006	Technische Universität Darmstadt, Germany Research assistant at the research group Peer-to-Peer Systems at the Multimedia Communications Lab (KOM)
10/2000–12/2006	Johanniter-Unfall-Hilfe, Dieburg, Germany Coordination of Patient Transport Ambulance
09/1999–10/2000	Johanniter-Unfall-Hilfe, Dieburg, Germany Civilian service

TEACHING ACTIVITY

Since WS 2008	Technische Universität Darmstadt, Germany Lecture and Exercise "Advanced Topics in Distributed Systems"
Since WS 2007	Technische Universität Darmstadt, Germany Seminar "Communication Systems and Multimedia: Advanced Topics of Future Internet Research"
Since SS 2007	Technische Universität Darmstadt, Germany Lecture and Exercise "Introduction to Net Centric Systems"
Since 2007	Technische Universität Darmstadt, Germany Tutor for various Bachelor- and Master theses

HONORS

8/2008	Best Paper Award at the IEEE International Workshop on Computational P2P Networks (COMP2P) for the paper: "*Understanding BitTorrent's Suitability in Various Applications and Environments*"

MEMBERSHIPS

Since 2008 IEEE student member

Darmstadt, den 25. November 2010

Sebastian Kaune

D

ERKLÄRUNG LAUT §9 DER PROMOTIONSORDNUNG

Ich versichere hiermit, dass ich die vorliegende Dissertation allein und nur unter Verwendung der angegebenen Literatur verfasst habe.

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, den 25. November 2010

Sebastian Kaune