RatFish: A File Sharing Protocol Provably Secure Against Rational Users

Michael Backes[1,2], Oana Ciobotaru[1], and Anton Krohmer[1]

[1] Saarland University [2] MPI-SWS

Abstract. The proliferation of P2P computing has recently been propelled by popular applications, most notably file sharing protocols such as BitTorrent. These protocols provide remarkable efficiency and scalability, as well as adaptivity to dynamic situations. However, none of them is secure against attacks from rational users, i.e., users that misuse the protocol if doing so increases their benefits (e.g., reduces download time or amount of upload capacity).
We propose a rigorous model of rational file sharing for both seeders and leechers, building upon the concept of rational cryptography. We design a novel file sharing protocol called RatFish, and we formally prove that no rational party has an incentive to deviate from RatFish; i.e., RatFish constitutes a Nash equilibrium. Compared to existing file sharing protocols such as BitTorrent, the tracker in RatFish is assigned additional tasks while the communication overhead of a RatFish client is kept to a minimum. We demonstrate by means of a prototype implementation that RatFish is practical and efficient.

1 Introduction

Recently, the peer-to-peer (P2P) paradigm has emerged as a decentralized way to share data and services among a network of loosely connected nodes. Characteristics such as failure resilience, scalability and adaptivity to dynamic situations have popularized P2P networks in both academia and industry. The proliferation of P2P computing has also been propelled by popular applications, most notably file sharing protocols such as BitTorrent [6].

A crucial assumption underlying the design of such file sharing protocols is that users follow the protocol as specified; i.e., they do not try to bypass the design choices in order to achieve higher download rates, or to avoid uploading to the system at all. However, not all users are necessarily altruistic, and publicly available, modified BitTorrent clients like BitThief [20] or BitTyrant [22] can be used to strategically exploit BitTorrent's design to achieve a higher download while contributing less or nothing at all in return. While several minor protocol adaptations have

been suggested to mitigate the attacks underlying these clients [30], the core weaknesses remain: In its current form, BitTorrent – and current file sharing protocols in general – offer better service to cheating clients, thereby creating incentives for users to deviate from the protocol; in turn, it further decreases the performance of honest clients. The task is thus to design a protocol that not only retains the remarkable characteristics of current file sharing protocols, but that is rational in the sense that it offers sufficient incentives for users to stick to the precise protocol specification. In more technical terms, this file sharing protocol should constitute an equilibrium state: Adhering to the protocol should optimize the benefits received by each individual participant, and any deviation from the protocol should result in a lower payoff for the cheating user.

1.1 Contributions

We contribute RatFish, a protocol for rational file sharing. RatFish is built upon the concepts and design choices that underlie BitTorrent, but it resolves the weaknesses that clients such as BitThief and BitTyrant exploit. We achieve this mostly by ensuring rational exchange of pieces between leechers and by having the tracker participate in the coordination of the downloads. In this context, an exchange is called rational if the participants have no incentive to deviate from it.

The distinctive feature of RatFish, however, is not that it discourages the use of several selfish strategies, but that it comes with a formal proof that deviating from RatFish is irrational for both seeders and leechers. In order to do this, we first characterize rational behaviors of leechers and seeders in file sharing protocols, building upon the concept of the recently emerging field of rational cryptography, in which users are defined as rational players in a game-theoretic sense. Intuitively, leechers are primarily interested in minimizing their download time and the amount of uploaded data, whereas seeders value the efficiency of the protocol in using their upload capacity. We cast this intuition into a rigorous mathematical model, and we formally prove that our protocol is secure against deviations of rational parties, by showing that it constitutes a Nash equilibrium. This holds even though RatFish allows users to dynamically leave and (re-)join. We prove this Nash equilibrium using a new proof technique that is of independent interest for rational cryptography: the step-by-step substitution of a deviating strategy with hybrid, semi-rational strategies.

We have built a prototype implementation of RatFish that demonstrates that RatFish is practical and efficient. We stress though that the

purpose of RatFish is not to achieve performance improvements over existing protocols, but to establish a formal proof that under realistic conditions, such as dynamically joining users, no rationally-behaving user has an incentive to deviate from RatFish. The additional computational overhead of RatFish compared to BitTorrent is small: basic cryptographic primitives (symmetric encryptions and digital signatures schemes) are used, and the tracker is assigned additional tasks such as the coordination of downloads and the generation of user incentives. The communication overhead of a RatFish client is kept to a minimum.

1.2 Related Work

The performance of BitTorrent has been thoroughly studied [25,3,12,24,23]. All these works attest to the impressive performance of BitTorrent in the presence of honest participants; however, it has been noted [3] that the rate-based Tit-For-Tat policy of BitTorrent does not prevent nodes from uploading less content than they should serve (in all fairness), thereby creating an incentive for abuse of the protocol.

The behavior of BitTorrent in the presence of cheating peers was subsequently investigated [19,22,20,27], revealing that cheating leads to a loss in overall performance for honest peers.

Our rigorous model of rational file sharing is grounded in the recently emerging field of rational cryptography, where users are assumed to only deviate from a protocol if doing so offers them an advantage. Rational cryptography is centered around (adapted) notions of game theory such as computational equilibria [7]. A comprehensive line of work already exists that develops novel protocols for important cryptographic primitives such as rational secret sharing and rational secure multiparty computation [8,11,10,1,15,9].

There has been already a variety of research aimed at making BitTorrent more robust against deviations of rationally-behaving users [30,16,23,21,28]. All these works provide stronger user incentives: they choke problematic connections [30], grant additional bandwidth to generously uploading neighbors [16], reward leechers that continue seeding after their download is completed [23], optimally distribute a seeder's bandwidth across swarms [21], and employ fair exchange protocols to stop leechers from aborting the protocol [28] early. These modified protocols, however, are still prone to rational attacks; in particular, none of these works reached a (Nash) equilibrium. There has been research [28] on how to ensure that certain deviations from selfish leechers or attacks

of malicious peers cannot succeed, e..g., no peer can assume another authorized peer's identity. However, so far there is no work which ensures that deviating from the protocol cannot yield better results.

There is also previous work that strived to establish an equilibrium in the context of file sharing [25]. However, this equilibrium was severely restricted in that it was only guaranteed when rational parties were allowed to only tweak the protocol parameters, but not when they could deviate in larger ways.

More recent research such as BAR-B [2], Equicast [14], and FOX [26] aimed at deploying incentives and punishments such that obeying the protocol is the best strategy for every rational player. The first two protocols were shown to be strict Nash equilibria, i.e., a rational peer obtains no benefit from unilaterally deviating from the assigned strategy. The drawback is that their strict equilibrium solutions limit the design: the BAR-B system only permits a static set of users. Equicast requires the rate at which leechers join to precisely match the rate of which they leave and considers only restricted utility functions that do not take downloading time into account; moreover, these protocols require nodes to waste network bandwidth by sending garbage data to balance bandwidth consumption. FOX [26] establishes a stable Nash equilibrium, but again it only allows a static set of leechers; moreover, its rationality is not grounded on incentives but on fear of retaliation such that a single Byzantine node can cause the entire system to collapse. Somewhat orthogonal to our work are the file streaming applications BAR-Gossip [18] and FlightPath [17]. Both works show a Nash equilibrium (a strict one for BAR-GOSSIP, and an approximate one for Flightpath), but rational players are only interested in minimizing the amount of uploaded data and reducing jitter. While such time-independent utility functions are reasonable for streaming applications, they do not apply to the more sophisticated setting of rational file sharing, where minimizing the time to complete a download is usually the primary goal. Moreover, none of these five protocols considers seeders as part of the rational model. We conclude by saying that like our approach, none of these works offers resistance against Sybil attacks. A Nash equilibrium ensures that no individual user has an incentive to deviate. However, it conceptually does not take coalitions of users into account, rendering Sybil attacks possible in most works on rationally secure protocols.

1.3 Outline

Section 2 provides a bird's eye view of the core ideas underlying how we create incentives in file sharing. Section 3 summarizes the concepts we use from rational cryptography and defines rational behaviors of seeders and leechers. Section 4 presents the RatFish protocol in detail. Section 5 contains the proof of equilibrium for RatFish; i.e., it shows that users cannot achieve a better payoff by deviating from the protocol. Section 6 discusses our experimental results. Section 7 concludes this work.

2 A Bird's Eye View on How to Rationalize P2P

For the sake of exposition, we provide a high-level overview of the core ideas underlying how we create incentives in file sharing. We briefly discuss which behaviors of seeders and leechers we consider rational, intuitively explain how to incentivize these behaviors, and finally discuss how an equilibrium is obtained for a small example protocol. In this section, we abstract away many important system's details and impose several assumptions to improve understanding. However, all these restrictions will be removed in section 4 where we present our RatFish protocol in its full generality.

In the following, we consider a single seeder S that intends to upload a file f to leechers $L_1, \ldots, L_M$. The file is split into pieces $f_1, \ldots, f_{M^2}$. In this exposition, we describe a simplistic protocol that proceeds in a sequence of $M + 1$ monolithic rounds. We assume that the seeder can upload exactly M pieces per round and that every leecher is able to upload and to download at least M pieces of the file in each round.

On the Rationality of Seeding. A seeder is a player that uploads without requesting reciprocation. Intuitively, it thus acts rationally if it uses its upload time and upload speed as efficiently as possible; i.e., for any fixed upload speed and time that the seeder spends within the system, the average download time for all leechers should be as small as possible. It is thus in the interest of the seeder to incentivize leechers to share parts of the file amongst each other as this increases the throughput of the whole system. As a consequence, the naive approach of uploading the whole file to an arbitrary leecher at once cannot yield a rationally secure protocol: This leecher may just complete the download and leave, causing some pieces of the file to be effectively lost from the system. Moreover, since there is only one seeder in this simplistic protocol and the number of leechers is known and does not change, there is no need for a third party,

i.e., a tracker. In the simplistic protocol, the seeder sends each leecher L_i in each round j the piece $f_{j \cdot M+i}$.

On the Rationality of Leechers. Leechers aim to download the file as fast as possible while saving upload capacity. The protocol thus has to enforce leecher participation as they will otherwise just download and leave. We need to propose a suitable piece selection algorithm and a piece exchange mechanism that prevents parties from cheating each other. In our example, piece selection is easy: In each round j a leecher L_i holds a piece $f_{j \cdot M+i}$ obtained from the seeder that no one else has. As the leecher can upload M pieces per round, he can exchange with the rest of the leechers their unique pieces. To ensure fair exchanges, leechers first exchange the pieces in encrypted form and subsequently send the corresponding decryption keys.

How an Equilibrium is Achieved. We conclude with some basic intuition on why no rational user has an incentive to deviate from the protocol. If all peers adhere to the protocol, the seeder will upload the file exactly once and stay in the system for M rounds. Each of the leechers will upload $M^2 - M$ pieces and complete its download after $M + 1$ rounds. It is easy to see that the average download time and hence the seeder's utility cannot be improved.

This outcome cannot be further improved for the leechers either: None of the leechers can download the file in less than $M + 1$ rounds since after round M each of them is missing at least $M - 1$ pieces. This holds because the protocol treats the rounds integrally. Otherwise, we could split a sufficiently big file into M^K pieces for some K and achieve a slightly reduced, optimal download time of $M + \frac{M^2}{M^K}$ using an analog algorithm. Moreover, since the seeder only provides M pieces to each of its peers, no leecher can obtain the full file without uploading at least $M^2 - M$ pieces in exchange for the pieces that it is missing from the seeder. This statement holds as no leecher can cheat during the rational piece exchange protocol: A leecher spends his upload capacity to receive an encrypted piece, hence he has no incentive not to send the much smaller decryption key to its partner. Thus, no party can improve its utility by deviating from the protocol.

3 A Game-theoretic Model for File Sharing

In this section, we propose a game-theoretic model for rationally secure file sharing. We start by reviewing central concepts from game theory and rational cryptography.

A *Bayesian game* $\Gamma = (\{T_i\}_{i=1}^n, \{A_i\}_{i=1}^n, Pr, \{u_i\}_{i=1}^n)$, also called *a game with incomplete information*, consists of *players* $1, \ldots, n$. The incomplete information is captured by the fact that the *type* for each player i (i.e., its private information) is chosen externally, from a set T_i, prior to the beginning of the game. Pr is a publicly known distribution over the types. Each player has a set A_i of possible *actions* to play and individual *utility functions* u_i. Actions are played either simultaneously or sequentially; afterwards, every player i receives a *payoff* that is determined by applying its utility function u_i to the vector of types received in the game, i.e., *profile types*, and the actions played, i.e., *action profile*.

Recent work has extended the traditional notion of a game to the requirements of cryptographic settings with their probabilistically generated actions and computationally-bounded running times. The resulting definition – called *computational game* [13] – allows each player i to decide on a probabilistic polynomial-time, in the security parameter, interactive Turing machine M_i (short PPITM). The machine M_i is called the *strategy* for player i. The output of M_i in the joint execution of these interactive Turing machines denotes the actions played by participant i.

Definition 1 (Computational Game). *Let k be the security parameter and let $\Gamma = (\{T_i\}_{i=1}^n, \{A_i\}_{i=1}^n, Pr, \{u_i\}_{i=1}^n)$ be a Bayesian game. Then Γ is a* computational game *if the played action A_i of each participant i is computed by a PPITM M_i and if the utility u_i of each player i is polynomial-time computable.*

Because of the probabilistic strategies, the utility functions u_i now correspond to the expected payoffs. Thus, when there is no possibility for confusion, we overload the notation for u_i. However, when the utility we employ is not clear from the context, we denote by U_i the expected utility for party i.

Rationally behaving players aim to maximize these payoffs. In particular, if a player knew which strategies the remaining players intend to choose, he would hence pick the strategy that induces the most benefit for him. As this simultaneously holds for every player, we are looking for a so-called *Nash equilibrium*, i.e., a strategy vector where each player has no incentive to deviate from, provided that the remaining strategies do not change. Similar to the notion of a game, we consider a computational variant of a Nash equilibrium.

Definition 2 (Computational Nash Equilibrium). *Let Γ be a computational game, where $\Gamma = (\{T_i\}_{i=1}^n, \{A_i\}_{i=1}^n, Pr, \{u_i\}_{i=1}^n)$ and let k be*

the security parameter. A strategy vector *(or* machine profile) consisting of PPITMs $\overrightarrow{M} = (M_1, \ldots, M_n)$ is a *computational Nash equilibrium* if for all i and any PPITM M_i' there exists a negligible function ϵ such that

$$u_i(k, M_i', \overrightarrow{M_{-i}}) - u_i(k, \overrightarrow{M}) \leq \epsilon(k)$$

holds.

Here $u_i(k, M_i', \overrightarrow{M_{-i}})$ denotes the function u_i applied to the setting where every player $j \neq i$ sticks to its designated strategy M_j and only player i deviates by choosing the strategy M_i'. In the definition above, we call M_i a *computational best response* to $\overrightarrow{M_{-i}}$.

We finally define the *outcome* of a computational game as the transcript of all players' inputs and the actions each has taken. In contrast to strategy vectors, an outcome thus constitutes a finished game where every player can determine its payoff directly. A utility function is thus naturally defined on the outcome of a computational game: When applied to a strategy vector with its probabilistic choices, it describes the vector's expected payoff; when applied to an outcome of the game, it describes the exact payoff for this outcome.

3.1 A Game-theoretic Model for File Sharing Protocols

We now define the utility functions for seeders and leechers such that these functions characterize rational behavior in a file sharing protocol. We start by introducing common notation and some preliminaries.

Notation and Preliminaries. Following the BitTorrent convention, we call a player in the file sharing game a *peer*. The peers are divided into two groups: A *seeder* uploads to other peers a *file* f that it owns, whereas a *leecher* downloads f. To mediate the communication among peers, we thus implicitly require a trusted party called the *tracker*. The tracker holds a signing key pair (pk, sk), and we assume that its IP address and public key pk are known to all peers.

The file f consists of pieces $f_1, \ldots, f_N$, each of length B bytes. The participants in the file sharing protocol hold the values $h_1 = h(f_1), \ldots, h_N = h(f_N)$, where h is a publicly known *hash function*. When deployed in practice, this publicly known information is distributed via a *metainfo file*. The tracker is only responsible for coordinating peers that are exchanging the same file. In order to stay close to a realistic setting, we allow different peers to have different upload and download capacities. Every seeder S_i has its individual *upload speed* $up_i^s(t, o)$ that depends on the

time t and the outcome o. Note that a seeder does not download anything except for metadata; hence we do not need to consider the download speed of seeders. Similarly, every leecher L_i has individual *upload* and *download speeds* $up_i^l(t,o)$ and $down_i^l(t,o)$. We denote by $T_{i,\text{fin}}(o)$ the total time that leecher L_i spends downloading the file. In terms of measurement units, each of the upload and download capacities defined so far is considered as bytes per second. Additionally, the time is measured in seconds. To increase readability, we omit the outcome in all formulas whenever it is clear from the context. We also introduce the sets $L = \{i \mid L_i \text{ is a leecher}\}$ and $S = \{i \mid S_i \text{ is a seeder}\}$.

Rationally-behaving Seeders. A seeder uploads parts of the file to other peers without requesting reciprocation. Intuitively, a seeder is interested in using as efficiently as possible its upload time and upload speed. Thus for any fixed upload speed and time that the seeder spends within the system, the average download time for all leechers should be as small as possible. We express this preference by the following seeder's utility function.

Definition 3 (Seeder's Utility Function). *We say that u_i^s is a* utility function for a seeder S_i *if for any two outcomes o, o' of the game with the same fixed upload speed up_i^s and fixed time T_i^s spent by S_i in the system, it holds that $u_i(o) \geq u_i(o')$ if and only if $\frac{1}{|L|}\sum_{i\in L} T_{i,\text{fin}}(o) \leq \frac{1}{|L|}\sum_{i\in L} T_{i,\text{fin}}(o')$.*

If S_i is the first seeder in the system, we implicitly require that S_i uploads the whole file at least once. Otherwise, it is not rational to share the file in the first place.

Rationally-behaving Leechers. Leechers aim at downloading the shared file as fast as possible; moreover, they also try to use as little upload capacity as possible. The relative weight of these two (typically contradictory) goals is given by a parameter α_i in the system measuring time units per capacity units, e.g., $sec^2/bytes$.

Definition 4 (Leecher's Utility Function). *Let $\alpha_i \geq 0$ be a fixed value. We say that u_i^l is a* utility function for leecher L_i *if the following condition holds: For two outcomes o, o', L_i prefers outcome o to o' if and only if*

$$\alpha_i \cdot \int_0^{T_{i,\text{fin}}(o)} up_i^l(t,o)\,dt + T_{i,\text{fin}}(o) \leq \alpha_i \cdot \int_0^{T_{i,\text{fin}}(o')} up_i^l(t,o')\,dt + T_{i,\text{fin}}(o').$$

The value α_i corresponds to L_i's individual valuation for upload speed and time; e.g., if $\alpha_i = 0.5\frac{sec^2}{bytes}$, the leecher values time twice as much as the uploaded data.

In particular, this definition implies that a rationally-behaving leecher prioritizes completing the download over everything else: If the leecher does not download the file in outcome o, then $T_{i,\text{fin}}(o)$ equals infinity. If it downloads the file in some outcome o', then $T_{i,\text{fin}}(o')$ is finite and thus increases its payoff.

4 The RatFish Protocol

We now present the RatFish protocol. We start with the description of the tracker and proceed with the seeders and leechers, respectively.

4.1 The Protocol of the Tracker

Similar to BitTorrent, our tracker manages all valid IP addresses in the system and introduces new leechers to a set of neighbors. However, we assign the tracker additional tasks: First, our tracker is responsible for awarding each newcomer with seeding capacity equivalent to γ file pieces, for a tunable parameter γ. In practice, γ is a small constant number just big enough for the new leecher to participate in the system. As long as the seeders can provide γ pieces to each newly joining leecher, this value does not influence the existence of the Nash equilibrium.

Second, our tracker keeps track of which file pieces each peer owns at any given moment. This bookkeeping will be crucial for incentivizing peers to follow the RatFish protocol, for computing the deserved rewards and for answering queries about the leechers' availabilities. Third, the tracker imposes a forced wait for every leecher upon connecting, thereby preventing leechers from gaining advantages by frequently disconnecting and rejoining the protocol. Finally, if a leecher wishes to disconnect, the tracker provides a certificate on the most recent set of pieces the leecher has to offer. This allows leechers to later reconnect to RatFish and use their partially downloaded data, i.e., in order to cope with network disruptions. In the following, we describe the individual subprotocols of the tracker in detail. A rigorous description is given in Fig. 1, Fig. 2, Fig. 3 and Fig. 4.

The Connect Protocol. The tracker assigns every new leecher L_i a random subset of size H of all leechers that are currently in the system. This random subset corresponds to L_i's local neighborhood. The tracker sends

this neighborhood information to L_i after T seconds. Once the forced wait is over, the leecher may start downloading γ free pieces from seeders. The rationale behind this forced wait is that granting newly joined leechers free pieces creates incentives for whitewashing, i.e., frequent disconnecting and rejoining. Intuitively, the forced wait is a simple defense against such a behavior. From a rational perspective, if a leecher joins the system only once, the induced small delay will not be hurtful; however, whitewashing by frequently rejoining will cause an accumulated delay that will result in a smaller payoff. The forced wait is achieved by having the tracker sign the leecher's connecting time and IP address. Such signed timestamps are exchanged between neighboors and are used to determine whether leechers are allowed to start uploading to eachother. Neighbors use the timestamps to determine whether they are allowed to start uploading to each other. Thus as long as a user's IP address does not change, it can idle and become active again without being penalized by a forced wait, since the user's old signature on its IP address and time is still valid. This is detailed in Fig. 1.

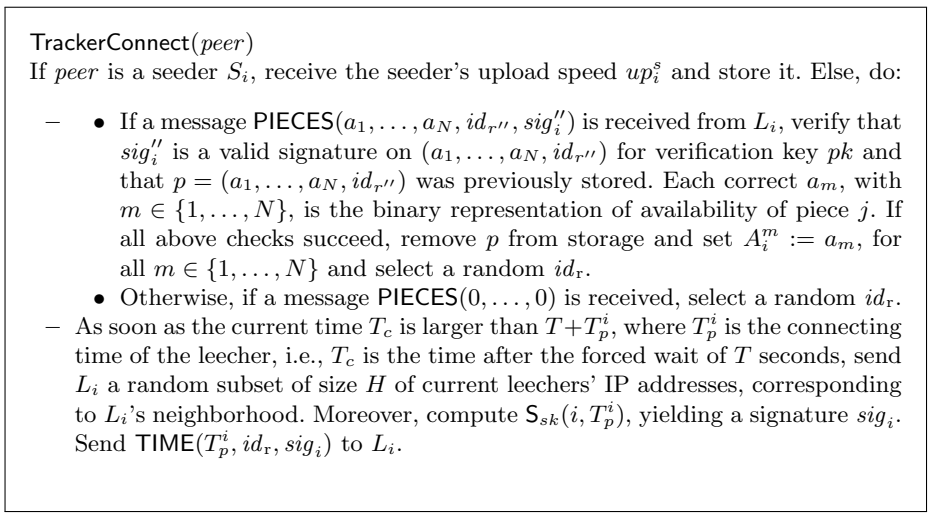

Fig. 1. The protocol of the tracker with procedure Connect.

The RatFish tracker has a mechanism for proving piece availability of rejoining leechers: it chooses a random rejoin ID id_r and signs it together with the departing leecher's piece availability. The tracker stores the availability status for leecher L_i and each piece m in the variable A_m^i.

The algorithms of the trackers ensure that $A_m^i = 1$ if L_i has m-th piece of file f and $A_m^i = 0$ otherwise. The leecher uses id_r to prove its piece availability to the tracker upon rejoining the system. The rejoining id_r is then deleted from the tracker's memory preventing leechers from reconnecting twice using the same reconnect id_r. This is detailed in Fig. 1, Fig. 2 and Fig. 3.

CheckExchange

Do the following steps unless one of their checks fails; abort in this case: (Below it is given only the protocol between the tracker and L_i. By symmetry, we can obtain the protocol between the tracker and L_j, where L_j is the exchange partner of L_i for the exchange described below.)

- Upon receiving a message $\mathsf{HAS}(j, y)$ from a leecher L_i, send back 1 if $A_j^y = 1$, and 0 otherwise.
- Upon receiving a message $\mathsf{EXCHANGED}(j, y, x)$ from a leecher L_i, indicating that pieces f_x and f_y have been exchanged with L_j, send the message $\mathsf{ACKNOWLEDGE}(i, x, y)$ to L_j.
- Upon subsequently receiving the message $\mathsf{OK}(i, x, y)$ from L_j, set $X_i := X_i + 1$ and $A_i^y := 1$, and send the message $\mathsf{OK}(j, y, x)$ to L_i. X_i denotes the number of acknowledged exchanges of leecher L_i in the current round of T seconds. X_i is reset to 0 and computed anew each round.

Fig. 2. The protocol of the tracker with procedure Check Exchange.

PeerLeave(i)

- Compute $sig_i' := \mathsf{S}_{sk}(A_i^1, \ldots, A_i^N, id_\mathrm{r})$. Store $(A_i^1, \ldots, A_i^N, id_\mathrm{r})$.
- Send the message $\mathsf{LEAVE}(id_\mathrm{r}, sig_i')$ to L_i and disconnect from L_i.

Fig. 3. The protocol of the tracker with procedure PeerLeave.

The Reward Protocol. The reward system constitutes the crucial part of RatFish. The underlying idea is to reward only leechers who are exchanging. We only allow one exception to this rule: The leechers that have just connected to the tracker in the previous round are also entitled

to a reward of γ pieces in the current round. Thus the seeders do not automatically upload to their neighborhood as in BitTorrent; rather they are told by the tracker whom to upload to.

To determine whether an exchange between L_i and L_j has indeed taken place, the tracker asks both L_i and L_j to acknowledge the exchange. If the acknowledgements succeed, the tracker internally increases the variables X_i and X_j, which corresponds to the number of file piece exchanges of L_i and L_j, respectively. The tracker moreover stores which pieces of the file the leechers now additionally know. This is detailed in Fig. 4. Details on the participation of the tracker in the exchange protocol are given in Sect. 4.3, where the rational exchange of pieces between leechers is explained.

RewardLeechers (called every T seconds, i.e., at the start of a new round)

- Award γ pieces to every leecher that joined in the previous round. Let *prev* be the number of these leechers.
- Compute for every leecher L_i its deserved percentage of seeders' upload speed:

$$r_i := \min\left\{\frac{X_i}{\sum_{k\in L} X_k}, \frac{1}{2}\right\}$$

- Let $\beta_i := r_i \cdot \left(\frac{\sum_{k\in S} up_k^s \cdot T}{B} - \gamma \cdot prev\right)$. For every i, assign a set of seeders to jointly offer β_i pieces to L_i such that the individual upload capacity of the seeders is respected, see Sect. 4.1. Send to every seeder the corresponding set of leechers and the number of pieces that these leechers should download from them.
- Set $A_i^y := 1$ when a seeder informs the tracker that f_y was downloaded by L_i. Reset $X_i := 0$ for all i.

Fig. 4. The protocol of the tracker with procedure RewardLeechers.

Every round, i.e., after T seconds, the actual rewards are given out. The tracker distributes the seeders' upstream proportional to the number of exchanges every leecher made in the previous round. Hence, the more exchanges a leecher completed in a certain round, the larger the reward computed for him by the tracker, and hence the more download capacity the leecher receives from the seeders. At the beginning of a round, once the reward deserved by each leecher is computed, the old values X_i are reset to 0 and computed anew, according to the exchanges performed in

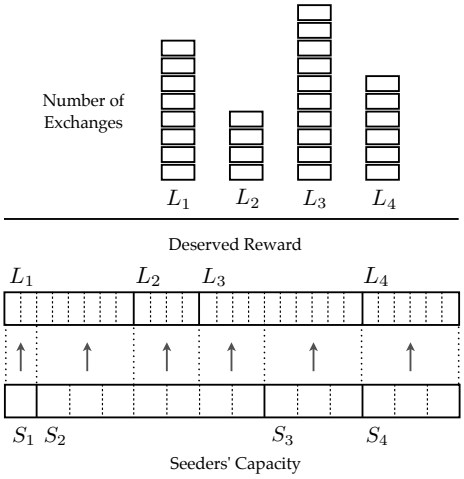

Fig. 5. Schematic distribution of the rewards

the current round. A graphical illustration of the reward protocol is given in Fig. 5.

More precisely, Fig. 5 depicts a system with a tracker, four leechers and four seeders. During the latest complete round of exchanges, leecher L_1 has performed a number of 8 exchanges and leechers L_2, L_3 and L_4 have performed a number of 4,10 and 6 exchanges, respectively. In terms of existing notation, this can simply be written as $X_1 = 8$, $X_2 = 4$, $X_3 = 10$ and $X_4 = 6$. This means that for example L_1 has performed $28,57\%$ of all exchanges during the latest complete round. Taking into account that there are no new leechers currently connecting to the system, according to Fig. 5, at the beginning of next round, L_1 will receive for free $28,57\%$ of all the seeding capacity currently available in the system. Given the existing upload capacity of the seeders, this effectively implies that the tracker informs seeders S_1 and S_2 that they should upload 2 and respectively 6 file pieces for free to L_1. It is assumed that S_1 and S_2 have the complete file, so L_1 can choose which pieces he wants to download for free. The same reward distribution algorithm applies for the rest of the leechers and seeders.

4.2 The Protocol of the Seeder

Upon connecting, the seeder informs the tracker about the upload speed it is going to offer. The tracker adds the seeder's speed to the amount of free available upload capacity. As the tracker performs all the computations

Seeding$_j$
Upon connecting, the seeder sends its upload speed up_j^s to the tracker. Additionally, in each round:

- Receive from the tracker a set M of leechers and the corresponding number of pieces ω_i that every leecher $L_i \in M$ should receive.
- Inform every leecher $L_i \in M$ how many pieces ω_i they are allowed to download.
- When a leecher $L_i \in M$ requests at most ω_i pieces by L_i (potentially incrementally in this round, i.e., it may ask for a few pieces first), send these pieces to L_i and send a message to the tracker that these pieces have been uploaded to L_i. Requests by leechers $L_j \notin M$ are ignored.

Fig. 6. The protocol of the seeder S_j.

for determining the rewards, the seeder simply proceeds by uploading the number of file pieces to the leechers as instructed by the tracker. To keep the tracker's information about individual leechers up-to-date, the seeder informs the tracker whenever it uploads a piece to a leecher. A rigorous description is given in Fig. 6.

4.3 The Protocol of the Leecher

From a rational perspective, the leecher protocol is the most difficult to get right: while the tracker is honest and seeders partially altruistic, a leecher tries to bypass the incentives for uploading wherever reasonable. Intuitively, the exchange protocol looks like this: Leechers use the signed messages from the tracker to verify each other's join times. Also, when two leechers exchange data, the tracker participates in this exchange: Before two leechers start an exchange, they verify with the tracker that the other party holds the desired piece. If this check succeeds, the encryptions of the pieces agreed upon are exchanged. Before they also send the key to each other to decrypt these messages, both leechers acknowledge the exchange to each other so that they get a higher reward.

The Connect Protocol. When a leecher connects to the tracker for the first time, it requests a local neighborhood. If the leecher rejoins, it additionally proves to the tracker that it already owns some pieces of the file by sending the signature received from the tracker at its last disconnect. When connecting to a seeder, the leecher requests pieces until its seeder's reward is depleted.

$\mathsf{LeecherConnect}_i(party)$
If *party* is the tracker, then:

1. If L_i rejoins the protocol, send $\mathsf{PIECES}(a_1, \ldots, a_N, id_\mathrm{r}, sig_i')$ to the tracker where $a_m = 1$ if L_i owns the m-th piece of the file, id_r is the rejoin ID and sig_i' is the signature received when disconnecting from the system last time. If L_i is a new leecher, it sends $\mathsf{PIECES}(0, \ldots, 0)$.
2. Receive $\mathsf{TIME}(T_p^i, id_\mathrm{r}, sig_i)$ from the tracker – indicating the signed connecting time and ID, as well as a set of neighbors' IP addresses. Connect to them.

If *party* is a leecher L_j, do (abort if a check fails):

- Send the message $\mathsf{MYTIME}(T_p^i, sig_i)$ to L_j.
- Receive the message $\mathsf{MYTIME}(T_p^j, sig_j)$ from L_j. Verify that sig_j is a valid signature on (j, T_p^j) for pk and that $T_c > T_p^j + T$ holds.
- Send $\mathsf{AVAILABILITY}(a_1, \ldots, a_N)$ to L_j where $a_m = 1$ if L_i owns f_m.
- Receive the message $\mathsf{AVAILABILITY}(a_1', \ldots, a_N')$ from L_j.

Fig. 7. The Connect protocol for leecher L_i.

Upon contacting another leecher, it waits until both forced waits are over. Afterwards, both leechers exchange information such that they know which pieces they can request from each other. To keep track of the availability in its neighborhood, the leecher observes the messages that leechers broadcast to their local neighborhood, indicating which pieces of the file they have just downloaded. A detailed description is given in Fig. 7.

The Piece Exchange. The piece exchange protocol run between two leechers uses encryptions to ensure that no leecher can get a piece without completing the exchange phase. From a practical perspective, it is important to note that the key sizes are small compared to a file piece size. Thus the communication and storage overhead induced by the keys and cryptographic operations is kept manageable. Leechers additionally query the tracker to ensure that their exchange partners own a file piece they need. Moreover, leechers want their exchanges to be counted and rewarded. Thus, after the encryptions are exchanged, each leecher prompts the tracker to ask the other leecher for an acknowledgement. Intuitively, there is no incentive to deviate in this step as they still lack the key from the other party. Once the acknowledgement step is successfully completed, both leechers exchange the keys. If a leecher deviates from any of these steps, it is blacklisted by the other leecher. We stress that blacklisting is not required for the security proof; it solely constitutes a

LeecherAwarded
Whenever L_i is informed by a seeder S_j that it can download ω_i pieces, request up to ω_i pieces from S_j (potentially incrementally in this round, i.e., L_i may ask for a few pieces first), and download these pieces.

$\mathsf{Exchange}_i(f_x, j, y)$
If any of the following checks fails, blacklist L_j and abort.

- Send the message $\mathsf{HAS}(j, y)$ to the tracker and wait for a positive answer represented by a bit b = 1.
- Choose a random key $k_{j,x}$ and compute $c_{j,x} \leftarrow \mathsf{E}(k_{j,x}, f_x)$.
- Send $c_{j,x}$ to L_j and wait for c_y from L_j.
- Perform the following two steps in parallel and proceed once both steps are completed:
  - Send $\mathsf{EXCHANGED}(j, x, y)$ to the tracker and wait for $\mathsf{OK}(j, x, y)$ as response
  - If receiving $\mathsf{ACKNOWLEDGE}(j, y, x)$ from the tracker, reply with $\mathsf{OK}(j, y, x)$.
- Send the key $k_{j,x}$ to L_j.
- Upon receiving k_y from L_j, retrieve $f'_y \leftarrow \mathsf{D}(k_y, c_y)$ and verify $h_y = h(f'_y)$.
- Broadcast to the local neighborhood that you now own the piece y.

Fig. 8. The Award and Exchange protocols for leecher L_i.

common technique in this setting to deal with malicious parties. A detailed description is given in Fig. 8. Fair exchange protocols have been used in prior work to incentivize peers to fairly exchange information [28]. A fair exchange intuitively means that either both parties obtain what they want or none of them obtains something. In contrast to [28], however, RatFish needs to neither periodically renew cryptographic keys, nor implement a non-repudiable complaint mechanism to allow parties to prove possible misbehaviors; instead it relies on short acknowledgment messages for each recipient and on collecting these messages to monitor the file completion for the participants. In fact, RatFish implements a weaker version of fair exchange, which is called rational exchange [5,29]. Intuitively, an exchange protocol is rational if the self-interested participating parties do not have an incentive to deviate. A schematic overview of the core part of the piece exchange protocol is provided in Fig. 9.

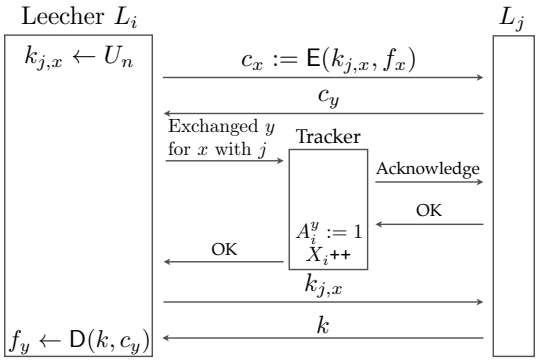

Fig. 9. The core part of piece exchange protocol between two leechers

5 Equilibrium Proof

In this section we prove that RatFish yields a computational Nash equilibrium; i.e., no leecher or seeder has an incentive to deviate from the protocol.

5.1 Underlying Assumptions

Recall that RatFish proceeds in rounds of T seconds. For simplicity, we assume that peers can join or leave only at the beginning or end of a round. This assumption can be easily enforced by letting the tracker force joining

peers to wait until the first round after at least T seconds pass. We also assume that it is impossible to forge identities on the IP layer (e.g., by using appropriate authentication mechanisms). Additionally, at least one seeder is present in the beginning to bootstrap the system and that the overall seeding capacity does not exceed twice the overall upload capacity of the leechers; this bound on the seeding capacity prevents the leechers from free riding, which is easy given enough seeding power. We moreover assume that each leecher's dedicated upload speed up_i^l is fully exhausted by other peers. This is equivalent to saying that there are enough leechers in the system such that each of them has enough neighbors for completely exhausting his upload capacity. These assumptions seem reasonable as the average seeders/leechers ratio is often close to 1:1 [4], and optimized centralized coordinators are capable of distributing upload capacities among different swarms [21]. Moreover, we assume that there exists a value δ such that for every leecher L_i the download speed $down_i^l$ is at most δ times larger than up_i^l. This assumption is not restrictive in the sense that we do not need to make any assumption regarding the magnitude of δ. Additionally, we assume keys do not contribute to the uploaded amount, since in practice, the size of the short keys is dominated by the size of the encrypted file piece. Moreover, we assume that each peer is only able to maintain one identity in the system. This in particular excludes Sybil attacks, in which multiple distinct identities are created by the same peer to subvert the reputation system of a P2P network. This assumption does not come as a surprise, since the Nash equilibrium conceptually does not defend against coalitions, rendering Sybil attacks possible in most works on rationally secure protocols. Regarding the cryptographic primitives, we assume that the signature scheme used by the tracker is secure against existential forgery under chosen-message attack and that the encryption scheme is semantically secure under chosen-plaintext attack. Finally, we assume that the encryption scheme is length preserving.

5.2 Proving the Nash Equilibrium

We finally show that RatFish constitutes a Nash equilibrium.

We first prove that a leecher deviating from the protocol cannot increase its utility by more than at most a negligible value, provided that no other party deviates. To show this, we determine two sets of possible *cheating actions* for leechers, which we call *independent actions* and *correlated actions*. Intuitively, the independent cheating actions can be *individually* replaced by honest actions without decreasing the utility, independent of the remaining parts of the deviating strategy. Correlated

cheating actions are sensitive to the details of the deviating strategy: we can only replace a correlated cheating action by a corresponding honest action without decreasing the utility if all deviating actions that jointly influence the leecher's utility are *simultaneously* replaced in one round. We show that the only correlated cheating action is the refusal of acknowledgement for an exchange.

Our proof technique starts with an arbitrary deviating strategy M_i' and provides a proof in two steps: In the first step, we replace all independent cheating actions step-by-step; here, a step within a strategy denotes the computation performed within the strategy between two consecutive outputs. Slightly more formally, let M_i be the honest strategy for leecher L_i, M_i' a deviating strategy, and $\{H^*_{ack,j}\}_j$ the set of all strategies that in every step are either honest or do not acknowledge an exchange. Then our technique takes as input M_i' and yields a so-called *semi-honest* strategy $M_i^* \in \{H^*_{ack,j}\}_j$ that for every input and random tape outputs in every step the same action as M_i' whenever possible, and plays honest otherwise. We then show that the semi-honest strategy M_i^* cannot yield a worse payoff than M_i'. The proof is based on the novel concept of hybrid concatenation of strategies.

We start by proving the following:

Lemma 1 (No Independent Cheating Actions of Leechers). *Let γ be the number of uploaded pieces a newly joined leecher is awarded. Let M_i' be a deviating strategy of L_i and let M_i^* be the semi-rational strategy as defined above. Then for $\alpha_i \in [0, \frac{T}{\delta \cdot \gamma \cdot B}]$, we have*

$$u_i(k, M_i', \boldsymbol{M}_{-i}) - u_i(k, M_i^*, \boldsymbol{M}_{-i}) \leq \epsilon(k),$$

for some negligible function ϵ.

Proof. We consider two main cases, depending on the number of steps of M_i'. If M_i' does not terminate in a finite number of steps, then the time $T_{i,\text{fin}}(o)$ the leecher L_i has to spend in the system is infinite. Thus any strategy, including M_i^*, cannot give a worse payoff. If M_i' runs in $N \in \mathbb{N}$ steps, then we construct N hybrid concatenations and prove that for all $n = 0, \ldots, N-1$, there exist a negligible function ϵ_i^n such that it holds

$$u_i(k, M_i' \|_n^s M_i^*, \boldsymbol{M}_{-i}) \leq u_i(k, M_i' \|_{n+1}^s M_i^*, \boldsymbol{M}_{-i}) + \epsilon_i^n(k), \qquad (1)$$

where by $M_i' \|_n^s M_i^*$ we denote the strategy M_i' with the last n steps replaced by last n steps of the rational strategy M_i^*.

In order to show this, we use induction on the steps of strategy M_i'. As there is no difference between the base case and the inductive step, we present below only the later. We examine all possible independent cheating actions which could have been performed by M_i' at step $n+1$. Intuitively, such independent cheating actions can be grouped into two main categories: related to the messages sent or related to the messages received. When a message is sent, there are three possible scenarios: sending no message, sending the correct message or sending a malformed message. When a message is received, there are also three possible scenarios: receiving no message, receiving the message and using it correctly for necessary computations or receiving the message and using it in a different way than described by the protocol.

In the following, we will concentrate on the case of messages which are sent. For the messages received, the proof follows a very similar approach.

It is clear that if at step $n+1$ the action performed is the action prescribed by M_i^*, then we have:

$$u_i(k, M_i' \|_n^s M_i^*, \boldsymbol{M}_{-i}) = u_i(k, M_i' \|_{n+1}^s M_i^*, \boldsymbol{M}_{-i}). \tag{2}$$

Below we give a detailed case distinction for the situation when the action is either to send no message or to send a malformed message.

Case 1: *M_i' does not announce the correct timestamp that it received from the tracker.*

Because L_i is required to provide a valid signature from the tracker on the timestamp, this deviation will be undetected only with a negligible probability, corresponding to the probability of constructing a selective forgery against the signature scheme. If L_i is caught cheating, then the other peers do not let L_i connect to them. Therefore, we obtain that (1) holds.

Case 2: *M_i' does not announce any timestamp that it received from the tracker.*

As the other leechers follow the protocol as prescribed, they will not connect to a neighbor which did not announce his timestamp received from the tracker. Thus, in this case, leecher L_i cannot complete the file download and (1) trivially holds.

Case 3: *M_i' connects to leechers before the penalty wait of T seconds is over.*

Other leechers will not reply to L_i unless it provides a valid signature on a timestamp $T_p^i < T_c - T$. Since an invalid signature can be produced only with negligible probability, we have that (1) holds.

Case 4: *M_i' connects to leechers after the penalty wait of T seconds is over.*

The more time leecher L_i waits to connect to others after his penalty wait is over, the more his utility decreases as compared to the case when he follows the prescribed strategy. So (1) is true.

Case 5: *M_i' does not connect to leechers even after the penalty wait of T seconds is over.*

If L_i does not connect to other leechers in his neighborhood or L_i does not reply to their connect requests, then L_i will not be able to download the file at all or his upload capacity will not be fully utilized as it is the case when he follows the protocol. In both situations, his benefit by performing this deviation decreases compared to the prescribed run of the protocol. More formally, (1) holds.

Case 6: *M_i' accepts connections from leechers whose penalty wait is not over.*

Because those leechers will not connect to L_i, this does not change the outcome of the game. Therefore, (1) trivially holds.

Case 7: *M_i' does not accept connections from at least one leecher in its own neighborhood, even though leecher's penalty wait is over.*

Such a deviation would slow down L_i as his upload capacity would not be fully exhausted. Thus, we have (1).

Case 8: *M_i' announces to hold more pieces than it actually does.*

There are two possible cases. Either L_i tries to convince the tracker upon connecting that it holds more pieces, or it has sent wrong "HAVE x" messages. In the first case, L_i has to provide a valid signature from the tracker on wrong data, which is possible only with negligible probability. In the second case, this will not affect anything until some other leecher requests a piece f_x from L_i that it does not own yet. Then, in the exchange step, the tracker will send the other leecher the message that L_i does not have f_x. The exchange will be canceled and this will not allow L_i to increase his utility, thus fulfilling (1).

Case 9: *M_i' announces to hold fewer pieces than it actually does.*

This does not increase its utility, because the amount of pieces is used by other leechers to determine whether they can exchange with L_i. Fewer parties are willing to exchange with L_i if it claims to have fewer pieces. This case also fulfills (1).

Case 10: *M_i' does not make any announcement on the pieces it has.* Such a deviation from L_i would only slow it down as the other leechers would not connect to L_i for performing new exchanges: L_i would potentially have no interesting pieces to offer. Hence, (1) holds.

Case 11: *When receiving an encrypted piece from a neighbor, M_i' does not send anything as a reply to the corresponding leecher.*

In this case, the utility of L_i would increase only if the leecher is able to guess the file piece from the encrypted message. Since the encryption scheme is semantically secure against chosen plaintext attack, the deviating leecher would succeed only with negligible probability so (1) is fulfilled.

Case 12: *In the exchange phase, M_i' sends a wrong encryption upon being requested to exchange piece f_x.*

Assume L_i does not hold piece f_x. Because other leechers only request what it announced with "HAVE x", this means it wrongly announced that it has too many pieces. This case is already discussed above. Therefore, L_i has the piece the other party requested and the other party expects a message of the length $|\mathsf{E}_k(f_x)|$. But then, L_i can also send the encryption of the requested piece, because sending a wrong encryption still requires it to upload exactly as much as uploading the right encryption. Thus, this deviation leaves L_i with (1).

Case 13: *M_i' requests an invalid reward from the tracker.*

By assumption, M_i' cannot send a message under the identity of another player (we assumed authentic channels). However, its request will not be acknowledged by any other party, as they are sticking to the protocol. Therefore, this does not increase its utility and (1) holds.

Case 14: *M_i' makes no request for reward, even though he is entitled to such a reward.* By performing this deviation, the overall download time for L_i will increase, thus trivially (1) holds.

Case 15: *In the exchange phase, M_i' does not send the key in the end.*

If the exchange reached the keys sending phase, since L_i's partner follows the protocol, it means that the two parties have the right piece for one another and they have already exchanged encryptions. By assumption, the key size does not increase the uploaded amount so if L_i sends the key it will not reduce his utility and (1) still holds.

Case 16: *M_i' reconnects after r rounds.*

Let o' be the outcome when L_i follows the reconnecting strategy $M_i' \|_n^s M_i^*$ and let o be the outcome when L_i follows $M_i' \|_{n+1}^s M_i^*$. Since the strategies of all other parties do not change and a leecher can reconnect only at the beginning of a new round, we have the following relation: $T_{i,\text{fin}}(o') = T_{i,\text{fin}}(o) + \tau$. The value τ is the additional time L_i spends in the system since for at least one round, L_i did not interact with any leecher. In a similar way, if U_i is the overall amount of uploaded data for outcome o, then the amount of uploaded data in o' is at most $U_i - \gamma \cdot B$, where

$\gamma \cdot B$ is the number of free awarded bytes for a (re-)joining leecher. As a reminder, B represents the bytes size of a file piece.

We observe that in outcome o', after the rejoin, the leecher is missing at least $T \cdot up_i^l$ bytes compared to o. By assumption we know that the leecher never has a download speed larger than $\delta \cdot up_i^l$. Hence, the time τ the leecher needs to additionally stay in the system is given by how fast he can download the missing data. This is at least:

$$\tau \geq \frac{T \cdot up_i^l}{\delta \cdot up_i^l} = \frac{T}{\delta}.$$

Proving (1) is equivalent to proving

$$T_{i,\text{fin}}(o) + \alpha_i \cdot U_i \leq T_{i,\text{fin}}(o) + \tau + \alpha_i \cdot (U_i - \gamma B).$$

This holds true if $\alpha_i \leq \frac{\tau}{\gamma \cdot B}$. The last inequality holds since by assumption $\alpha_i \in [0, \frac{T}{\delta \cdot \gamma \cdot B}]$ and $\tau \geq \frac{T}{\delta}$.

Case 17: *M_i' sends incompliant messages.*

Because such messages are ignored by other parties, this does not affect the utility, and (1) is fulfilled.

To summarize, we obtain that (1) holds for all $n \in \{0, \ldots, N-1\}$. By summating al these equations and taking into account that N is polynomial in k and also that the sum of polynomially many negligible functions is still negligible, we infer that there exists a negligible function ϵ' such that:

$$\begin{aligned} u_i(k, M_i', \boldsymbol{M}_{-i}) &= u_i(k, M_i' \|_0^s M_i^*, \boldsymbol{M}_{-i}) \\ &\leq u_i(k, M_i' \|_N^s M_i^*, \boldsymbol{M}_{-i}) + \epsilon'(k) \\ &= u_i(k, M_i^*, \boldsymbol{M}_{-i}) + \epsilon'(k). \end{aligned}$$

This concludes our proof.

Thus far, we have transformed a deviating strategy M_i' into a semi-rational strategy M_i^* that uses only correlated cheating actions and does not decrease the payoff. In the second step, we replace all correlated cheating actions round-by-round until we reach the honest strategy M_i. We use a hybrid argument based on the hybrid concatenation of strategies to show that the honest strategy outperforms the semi-rational strategy for leechers.

Lemma 2 (No Correlated Cheating Actions of Leechers). *Let M_i be the honest strategy for L_i, i.e., following the RatFish protocol and let M_i^* be the semi-rational strategy as defined above. Then*

$$u_i(k, M_i^*, \boldsymbol{M}_{-i}) - u_i(k, M_i, \boldsymbol{M}_{-i}) \leq \epsilon(k),$$

holds for some negligible function ϵ.

Proof. We again use the hybrid concatenation of strategies. However, now we need to replace correlated deviations with honest actions; hence the hybrids are constructed over rounds, and the notation $A\|_n^r B$ denotes that the last n rounds of strategy A are replaced by the last n rounds of strategy B. The proof starts with the last round and replaces the deviating strategy with the RatFish protocol round-by-round. We show by induction that each such replacement will not decrease the utility, i.e., for every $n \in \{1, \ldots, R\}$, where R is the number of rounds for strategy M_i^*, there exists a negligible function ϵ_i^n such that:

$$u_i(k, M_i^*\|_{n-1}^r M_i, \boldsymbol{M}_{-i}) \leq u_i(k, M_i^*\|_n^r M_i, \boldsymbol{M}_{-i}) + \epsilon_i^n(k). \tag{3}$$

As a consequence, the payoff for the honest M_i cannot be exceeded.

For the base step, we examine the initial hybrid where no deviation has been removed. Clearly, it holds that $u_i(k, M_i^*\|_0^r M_i, \boldsymbol{M}_{-i}) = u_i(k, M_i^*, \boldsymbol{M}_{-i})$. For the inductive step, we have the induction hypothesis

$$u_i(k, M_i^*\|_{n-1}^r M_i, \boldsymbol{M}_{-i}) \leq u_i(k, M_i^*\|_n^r M_i, \boldsymbol{M}_{-i}) + \epsilon_i^n(k),$$

for some negligible function ϵ_i^n.

Now we consider the n-th hybrid: the last n rounds are all played according to M_i and all previous rounds are played according to M_i^*. If we compare this hybrid with the $n+1$st hybrid, the only difference is that the possibly deviating $n+1$-st round is replaced by an honest one. By definition, in round $n+1$, the strategy M_i^* says that L_i does not acknowledge $Z \geq 0$ correct exchanges, but his exchange partners acknowledge all exchanges made so far. If $Z = 0$, then M_i^* played in round n the honest strategy M_i, thus:

$$u_i(k, M_i^*\|_n^r M_i, \boldsymbol{M}_{-i}) = u_i(k, M_i^*\|_{n+1}^r M_i, \boldsymbol{M}_{-i}).$$

For the case where $Z > 0$, we investigate the properties of the utility function. To reach the phase of acknowledging the other's exchange, L_i first needs to upload the encrypted data. Therefore, the amount of uploaded data stays the same both in the deviating strategy and in the

honest strategy. The only possibility to increase the utility is then to download more data in the same time span. In the following, we show this is not possible.

With M_i, the leecher's amount of downloaded data corresponding to one round of T seconds equals

$$B \cdot X_i + T \cdot (\sum_{k \in S} up_k^s) \cdot \min \left\{ \frac{X_i}{\sum_{k \in L} X_k}, \frac{1}{2} \right\}.$$

Note that in the second summand, we have the reward gained in the next round, because the tracker rewards exchanges always one round later. Furthermore, the tracker caps the reward to at most one half of the total seeders' upload, according to the protocol in Fig. 2. However, for a non-deviating player, it is impossible to obtain more than half of the overall exchanges, thus we have $\min \left\{ \frac{X_i}{\sum_{k \in L} X_k}, \frac{1}{2} \right\} = \frac{X_i}{\sum_{k \in L} X_k}$. Using M_i^* in round $n+1$, the leecher gains $Z \cdot B$ bytes less from the cheated exchanges, but he may obtain higher reward from the seeders.

This sums up to at most

$$B \cdot (X_i - Z) + T \cdot (\sum_{k \in S} up_k^s) \cdot \min \left\{ \frac{X_i}{(\sum_{k \in L} X_k) - Z}, \frac{1}{2} \right\}.$$

Indeed, the intuition for the first summand is that when leecher L_i does not acknowledge Z exchanges in a round, there is only a negligible probability that L_i will obtain the corresponding decryption keys for these exchanges. This holds as L_i has only honest exchange partners and they respond to a non-acknowledge of a correct exchange by blacklisting the deviating L_i and also by interrupting communication with L_i. The second summand gives the amount of upload reward that L_i receives from the seeders in a round of T seconds when he does not acknowledge Z exchanges. The value $\frac{X_i}{(\sum_{k \in L} X_k) - Z}$ gives the ratio of acknowledged exchanges for L_i to the total number of acknowledged exchanges. Therefore, his utility will not increase if

$$Z \cdot B + T \cdot (\sum_k up_k^s) \frac{X_i}{\sum_k X_k} \geq T \cdot (\sum_k up_k^s) \cdot \min \left\{ \frac{X_i}{(\sum_k X_k) - Z}, \frac{1}{2} \right\}. \tag{4}$$

To simplify the inequality more, we can express the number of exchanges X_k as the ratio of capacity used for uploading in one correct round divided by the size of a file piece. Formally, we have

$$X_k = \frac{T}{B} \cdot up_k^l, \tag{5}$$

for all $k \in L$.

Indeed, due to our assumption, for each leecher L_k there are enough neighbors in the system that can exhaust L_k's upload capacity. Due to the fact that values X_k, with $k \in \{1, \ldots, n\}$, are defined for the case when everybody is honest, and following the protocol implies exchanging with the neighbors until the full upload capacity is exhausted, we have that (5) holds.

We are now ready to distinguish two cases for the current proof. In the first case, the deviating leecher increases the benefit from the seeders, but stays below $\frac{1}{2}$ of the overall seeding capacity. In the second case, the reward is capped by the tracker at exactly half of this capacity. Note that those computations below rely on the assumption that seeding power is at most twice as large as leeching power.

Case 1: $\min\{\frac{X_i}{(\sum_k X_k)-Z}, \frac{1}{2}\} = \frac{X_i}{(\sum_k X_k)-Z}$. Then (4) evaluates to

$$\frac{Z \cdot B \cdot (\sum_k X_k) \cdot ((\sum_k X_k) - Z)}{T \cdot (\sum_k up_k^s)} \geq Z \cdot X_i$$
$$\Leftrightarrow \underbrace{\frac{(\sum_k up_k^l)}{(\sum_k up_k^s)}}_{\geq \frac{1}{2}} ((\sum_k X_k) - Z) \geq X_i.$$

The last inequality holds as we are in case 1.

Case 2: $\min\{\frac{X_i}{(\sum_k X_k)-Z}, \frac{1}{2}\} = \frac{1}{2}$, then (4) evaluates to

$$Z \cdot B + T \cdot (\sum_k up_k^s) \frac{X_i}{\sum_k X_k} \geq \frac{T}{2} (\sum_k up_k^s)$$
$$\Leftrightarrow \underbrace{\frac{\sum_k up_k^l}{\sum_k up_k^s}}_{\geq \frac{1}{2}} Z \geq \frac{1}{2} (\sum_k X_k) - X_i.$$

The above inequality is fulfilled as we are in case 2. It follows that

$$u_i(k, M_i^* \|_n^r M_i, \boldsymbol{M}_{-i}) \leq u_i(k, M_i^* \|_{n+1}^r M_i, \boldsymbol{M}_{-i}) + \epsilon_i^{n+1}(k)$$

and this also concludes our induction proof. By summing up all the inequalities described by (3), where $n \in \{1, \ldots, R\}$ and taking into account that R is the number of rounds of a polynomially bounded strategy, we obtain there exists a negligible function ϵ' such that

$$\begin{aligned} u_i(k, M_i^*, \boldsymbol{M}_{-i}) &= u_i(k, M_i^* \|_0^r M_i, \boldsymbol{M}_{-i}) \\ &\leq u_i(k, M_i^* \|_R^r M_i^*, \boldsymbol{M}_{-i}) + \epsilon'(k) \\ &= u_i(k, \boldsymbol{M}) + \epsilon'(k). \end{aligned}$$

This concludes our proof.

Showing that seeders have no incentive to deviate from the protocol is considerably easier than the corresponding statement for leechers, since seeders are considered partially altruistic. We show that as long as all leechers follow the protocol, a seeder cannot run a deviating strategy to improve its payoff.

Lemma 3 (No Seeder Deviation). *There is no deviating strategy for any seeder that increases its payoff if all other parties follow the RatFish protocol.*

Proof. We prove that no matter how a seeder distributes its upload speed over the leechers, their average time to completion does not decrease. In particular, it implies the seeder's utility does not increase if its upload speed initially announced to the tracker decreases.

The statement holds as all other participants stick to their strategy: a seeder may deviate only by assigning leechers different weights than those received from the tracker. Let r be the last round in which the seeder S_j deviates and let up_i^s be its full upload speed. Let $\omega_1, \ldots, \omega_n$ be the weights on the upload speed to all leechers given by the tracker and let $\phi_1, \ldots, \phi_n$ be some other arbitrary weights. It holds that $\sum_{k=1}^{n} \phi_k \leq \sum_{k=1}^{n} \omega_k = 1$ and that we have $\omega_k, \phi_k \geq 0$ for all k. We obtain the following bound for the average download speed of all leechers in round r:

$$\begin{aligned}\frac{1}{|L|}\sum_{k\in L}(down_k^l + \omega_k \cdot up_i^s) &= \frac{1}{|L|}\left(\sum_{k\in L}(down_k^l) + up_i^s\right) \\ &\leq \frac{1}{|L|}\sum_{k\in L}(down_k^l + \phi_k \cdot up_i^s).\end{aligned}$$

Thus, if a seeder deviates from RatFish, it causes a decrease in the average download speed in round r for the leechers. Consequently, the average completion time for a leecher does increase when the seeder deviates, i.e., the seeder has no incentive to deviate from RatFish.

We finally combine the results that neither leechers nor seeders have an incentive to deviate (the tracker is honest by assumption) to establish our main result.

Theorem 1 (Computational Nash Equilibrium). *The RatFish protocol constitutes a computational Nash equilibrium if* $\alpha_i \in [0, \frac{T}{\delta \cdot \gamma \cdot B}]$ *for all* $i \in L$.

Proof. We show that for every participant in the protocol, a deviation increases the utility by at most a negligible value. Assume leecher L_i deviates from the strategy M_i described by the protocol using M_i'. Then, by Lemma 1, we have that there exists a sem-rational strategy $M_i^* \in \{H_{ack,j}^*\}_j$ such that for some negligible function ϵ_1 it holds

$$u_i(k, M_i', \boldsymbol{M}_{-i}) - u_i(k, M_i^*, \boldsymbol{M}_{-i}) \leq \epsilon_1(k).$$

However, by Lemma 2, we have that for all M_i^* and for some negligible function ϵ_2,

$$u_i(k, M_i^*, \boldsymbol{M}_{-i}) - u_i(k, \boldsymbol{M}) \leq \epsilon_2(k).$$

Therefore, we obtain that for all deviating strategies M_i' there exists a negligible function ϵ such that

$$u_i(k, M_i', \boldsymbol{M}_{-i}) - u_i(k, \boldsymbol{M}) \leq \epsilon(k).$$

Lemma 3 gives us an analogous result for seeders. Therefore, RatFish gives a computational Nash equilibrium.

After we have shown how to derive by hand the rather long and intricate proof that RatFish represents a Nash equilibrium, an important observation is due. First, it is not clear at first glance that our case analysis technique is exhaustive. Unfortunately, since the protocol is rather complex in terms of number of different messages which are sent and received and also in terms of the side effects that they may incur, it is not clear how to perform an exhaustive case analysis by hand. Second, using a tool or a method off-the-shelf that performs an automatic verification of the game theoretic property, namely computational Nash equilibrium, for a protocol that involves cryptographic primitives and assumptions, has not been, to the best of our knowledge, attempted yet. Thus, in the current work we perform an analysis using only known or adapted proofs techniques and we leave the automated analysis for future work.

6 Implementation and Performance Evaluation

In this section, we describe the implementation of RatFish and we experimentally evaluate its performance. We focus on the implementation and performance evaluation of the tracker, since the tracker took on several additional responsibilities and is now involved in every exchange. In contrast to the tracker, seeders and leechers are largely unchanged when compared to BitTorrent: the exchange of encrypted pieces constitutes a

small computational overhead, but leaves the network complexity that usually constitutes the performance bottleneck of P2P protocols essentially unchanged.

6.1 Implementation

The RatFish tracker was implemented using about 5000 lines of code in Java, thus ensuring compatibility with common operating systems. The implementation is designed to work with both UDP and TCP.

The messages sent in the protocol start with the protocol version number and message ID (which determines the length of the message), followed by the torrent ID, and additional information that depends on the type of message.

Afterwards, a task is created that processes the received message. This task is given to the threadpool executor – the main part of the RatFish tracker that also ensures parallelization. The threadpool sustains eight parallel threads and assigns new tasks to the next free thread. For instance, when the tracker receives a notification that a leecher joined the protocol, the task extracts the leecher's IP from this message and triggers the forced wait. After $T = 300$ seconds it replies with a digital signature for the leecher using an RSA-based signature scheme that signs SHA-1 hashes.

6.2 Experimental Setup

For the evaluation, we ran the RatFish tracker on a server with a 2-cores Intel Xeon CPU, 2GB of RAM, a 100MBit Internet connection and an Ubuntu SMP operating system with kernel version 2.6.28-18. We simulated a swarm with up to 50,000 peers, divided into neighborhoods of size 4. The simulated leechers send the same messages as a real leecher would, thus yielding an accurate workload measure for the tracker. Every leecher was configured to complete one exchange per second, and we chose the size of a piece to be 256 kB according to BitTorrent standards. Hence every leecher has a virtual upload speed of 256 kB/s. The size of the shared file is 50 MB, and the seeders upload one forth of the file per second in a round-robin fashion to their neighborhoods. The simulated clients are running on a separate machine. This allows us to measure network throughput. In our simulation, we need to pretend to the tracker that clients connect from different IPs. We thus used UDP in our experiments. Deploying RatFish in reality would presumably be based on TCP, which would slightly increase the network complexity.

6.3 Performance Evaluations

Fig. 10 depicts the results for our experiments. The main observation, shown in the left part of Fig. 10, is that even though we engage the tracker in every exchange in the swarm, the protocol scales well (a resource usage of 65% for 50,000 leechers). One can also observe that the computation complexity becomes a limiting factor, but we expect this to change for more cores given our highly parallelized implementation. Memory was not a bottleneck in any experiment.

The right part of Fig. 10 considers the case where many leechers join at once, but no exchanges are happening. This study is important since the tracker's most expensive task is to sign the join time of leechers. In our experiments, the tracker was able to serve about 400 new leechers per second. Since the server has T seconds for signing in practical deployments, the signature computation would be scheduled in free CPU time and hence not delay ongoing exchanges. We also observed that the two measurements depicted in Fig. 10 on CPU usage are additive, e.g., a system with 30,000 leechers and 200 joining leechers per second uses 90% of the CPU.

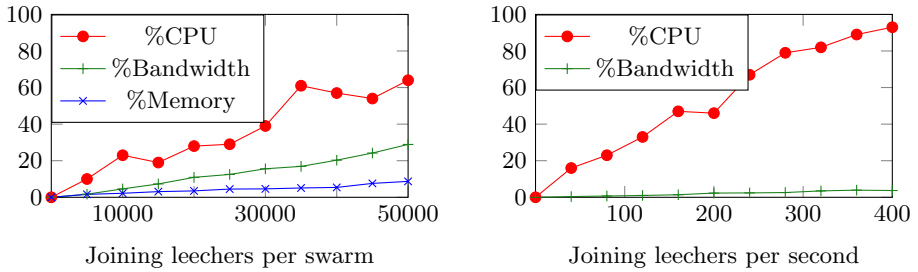

Fig. 10. Left: Resource usage for a static number of leechers engaging in exchanges. **Right:** Resource usage for dynamically joining leechers.

7 Conclusions and Future Work

We have proposed a file sharing protocol called RatFish that we have proven secure against deviations of rational users. We first characterized rational behaviors of leechers and seeders in file sharing protocols. Subsequently, we formally showed that no rational party has an incentive to deviate from RatFish; i.e., RatFish constitutes a Nash equilibrium. While the tracker in RatFish is assigned additional tasks compared to existing

file sharing protocols such as BitTorrent, the communication overhead of a RatFish client compared to a BitTorrent client is minimal. We have demonstrated by means of a prototype implementation that RatFish is practical and efficient.

A central question for future work on rational file sharing – and for rational cryptography in general – is whether the Nash equilibrium is a strong enough notion for real-world applications and threat models. Robustness against user coalitions would be more desirable. (See the discussion in [8] and [1].) RatFish already provides suitable hooks for potential mitigation techniques against coalitions, e.g., by ensuring that players entering small coalitions can only increase their utilities by a negligible amount; hence entering a coalition would be irrational in the first place. Moreover, RatFish currently considers file sharing for independent swarms only, i.e., seeders in one swarm cannot be leechers in another swarm. Extending RatFish to cope with such more a general setting requires to generalize the seeders' utility functions and to adapt the relevant parts of RatFish in order to maintain the Nash equilibrium property.

References

1. Abraham, I., Dolev, D., Gonen, R., Halpern, J.: Distributed computing meets game theory: robust mechanisms for rational secret sharing and multiparty computation. In: 25th Annual ACM Symposium on Principles of Distributed Computing (PODC'06). pp. 53–62. ACM (2006)
2. Aiyer, A.S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.P., Porth, C.: BAR fault tolerance for cooperative services. Operating Systems Review 39(5), 45–58 (2005)
3. Bharambe, A.R., Herley, C., Padmanabhan, V.N.: Analyzing and improving a BitTorrent network's performance mechanisms. In: The 25th IEEE Conference on Computer Communications (INFOCOM'06). pp. 1–12. IEEE (2006)
4. Bieber, J., Kenney, M., Torre, N., Cox, L.P.: An empirical study of seeders in BitTorrent. Tech. rep., Duke University (2006)
5. Buttyán, L., Hubaux, J.P.: Rational exchange - a formal model based on game theory. In: Electronic Commerce, Second International Workshop (WELCOM'01). pp. 114–126. Springer (2001)
6. Cohen, B.: Incentives build robustness in BitTorrent. Tech. rep., bittorrent.org (2003)
7. Dodis, Y., Halevi, S., Rabin, T.: A cryptographic solution to a game theoretic problem. In: 20th Annual International Cryptology Conference (CRYPTO'00). pp. 112–130. Springer (2000)
8. Feigenbaum, J., Shenker, S.: Distributed algorithmic mechanism design: recent results and future directions. In: 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M'02). pp. 1–13. ACM (2002)
9. Fuchsbauer, G., Katz, J., Naccache, D.: Efficient rational secret sharing in standard communication networks. In: 7th Theory of Cryptography Conference (TCC'10). pp. 419–436. Springer (2010)

10. Gordon, D., Katz, J.: Rational secret sharing, revisited. In: 5th Conference on Security and Cryptography for Networks (SCN'06). pp. 229–241. Springer (2006)
11. Halpern, J., Teague, V.: Rational secret sharing and multiparty computation: extended abstract. In: 36th Annual ACM Symposium on Theory of Computing (STOC'04). pp. 623–632. ACM (2004)
12. Izal, M., Uroy-Keller, G., Biersack, E., Felber, P.A., Hamra, A.A., Garces-Erice, L.: Dissecting BitTorrent: Five months in torrent's lifetime. In: Passive and Active Measurements (PAM'04). pp. 1–11. Springer (2004)
13. Katz, J.: Bridging game theory and cryptography: Recent results and future directions. In: 5th Theory of Cryptography Conference (TCC'08). pp. 251–272. Springer (2008)
14. Keidar, I., Melamed, R., Orda, A.: Equicast: Scalable multicast with selfish users. Computer Networks 53(13), 2373–2386 (2009)
15. Kol, G., Naor, M.: Cryptography and game theory: Designing protocols for exchanging information. In: 5th Theory of Cryptography Conference (TCC'08). pp. 320–339. Springer (2008)
16. Levin, D., LaCurts, K., Spring, N., Bhattacharjee, B.: BitTorrent is an auction: analyzing and improving BitTorrent's incentives. Computer Communications Review (CCR) 38(4), 243–254 (2008)
17. Li, H.C., Clement, A., Marchetti, M., Kapritsos, M., Robison, L., Alvisi, L., Dahlin, M.: FlightPath: Obedience vs. choice in cooperative services. In: 8th USENIX Symposium on Operating Systems Design and Implementation (USENIX OSDI'08). pp. 355–368. USENIX Association (2008)
18. Li, H.C., Clement, A., Wong, E.L., Napper, J., Roy, I., Alvisi, L., Dahlin, M.: BAR gossip. In: 7th Symposium on Operating Systems Design and Implementation (USENIX OSDI'06). pp. 191–204. USENIX Association (2006)
19. Liogkas, N., Nelson, R., Kohler, E., Zhang, L.: Exploiting BitTorrent for fun (not for profit). In: 5th International Workshop on Peer-to-Peer Systems (IPTPS'06) (2006)
20. Locher, T., Moor, P., Schmid, S., Wattenhofer, R.: Free riding in BitTorrent is cheap. In: 5th Workshop on Hot Topics in Networks (HotNets'06). pp. 85–90. ACM (2006)
21. Peterson, R.S., Sirer, E.G.: Antfarm: efficient content distribution with managed swarms. In: 6th USENIX Symposium on Networked Systems Design and Implementation (USENIX NSDI'09). pp. 107–122. USENIX Association (2009)
22. Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., Venkataramani, A.: Do incentives build robustness in BitTorrent? In: 4th Symposium on Networked Systems Design and Implementation (USENIX NSDI'07). pp. 1–14. USENIX Association (2007)
23. Piatek, M., Isdal, T., Krishnamurthy, A., Anderson, T.: One hop reputations for peer to peer file sharing workloads. In: 5th Symposium on Networked Systems Design & Implementation (USENIX NSDI'08). pp. 1–14. USENIX Association (2008)
24. Pouwelse, J.A., Garbacki, P., Epema, D., Sips, H.J.: The BitTorrent p2p file-sharing system: Measurements and analysis. In: 4th International Workshop on Peer-to-Peer Systems (IPTPS'05). pp. 205–216. Springer (2005)
25. Qiu, D., Srikant, R.: Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In: Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'04). pp. 367–378. ACM (2004)

26. Rob, D.L., Sherwood, R., Bhattacharjee, B.: Fair file swarming with FOX. In: 5th International Workshop on Peer-to-Peer Systems (IPTPS'06) (2006)
27. Shneidman, J., Parkes, D., Massoulié, L.: Faithfulness in internet algorithms. In: Workshop on Practice and Theory of Incentives and Game Theory in Networked Systems (PINS'04). pp. 220–227. ACM (2004)
28. Sirivianos, M., Yang, X., Jarecki, S.: Robust and efficient incentives for cooperative content distribution. Transactions On Networking 17(6), 1766–1779 (2009)
29. Syverson, P.F.: Weakly secret bit commitment: Applications to lotteries and fair exchange. In: Computer Security Foundations Workshop (CSFW'98). pp. 2–13. IEEE Computer Society (1998)
30. Thommes, R., Coates, M.: BitTorrent fairness: Analysis and improvements. In: 4th IEEE Workshop on the Internet, Telecommunications and Signal Processing (WITSP'05). IEEE (2005)