Design Space Analysis for Modeling Incentives in Distributed Systems

Rameez Rahman
Delft University of Technology
Delft, The Netherlands
rrameez@gmail.com

Tamás Vinkó
Delft University of Technology
Delft, The Netherlands
tamas.vinko@gmail.com

David Hales
The Open University
Milton Keynes, UK
david@davidhales.com

Johan Pouwelse
Delft University of Technology
Delft, The Netherlands
peer2peer@gmail.com

Henk Sips
Delft University of Technology
Delft, The Netherlands
h.j.sips@tudelft.nl

ABSTRACT

Distributed systems without a central authority, such as peer-to-peer (P2P) systems, employ incentives to encourage nodes to follow the prescribed protocol. Game-theoretic analysis is often used to evaluate incentives in such systems. However, most game-theoretic analyses of distributed systems do not adequately model the repeated interactions of nodes inherent in such systems. We present a game-theoretic analysis of a popular P2P protocol, BitTorrent, that models the repeated interactions in such protocols. We also note that an analytical approach for modeling incentives is often infeasible given the complicated nature of most deployed protocols. In order to comprehensively model incentives in complex protocols, we propose a simulation-based method, which we call *Design Space Analysis* (DSA). DSA provides a tractable analysis of competing protocol variants within a detailed design space. We apply DSA to P2P file swarming systems. With extensive simulations we analyze a wide-range of protocol variants and gain insights into their robustness and performance. To validate these results and to demonstrate the efficacy of DSA, we modify an instrumented BitTorrent client and evaluate protocols discovered using DSA. We show that they yield higher system performance and robustness relative to the reference implementation.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems – Distributed applications; H.1.0 [**Information Systems**]: Models and Principles – General; J.4 [**Computer Applications**]: Social and Behavioral Sciences – Economics

General Terms

Algorithms, Design, Economics, Theory

Keywords

Incentive systems, game theory, design space analysis, robustness

1. INTRODUCTION

Incentives play an important role in distributed systems with no centralized authority. Proper incentives ensure that the prescribed protocol is followed by all the nodes, i.e., the protocol is robust to strategic manipulation. A powerful tool for modeling incentives is game theory, the branch of economics that can model individual behavior in strategic situations [22]. The general applicability and predictive powers of game theory has allowed designers to employ it in a variety of contexts for the design of distributed systems [3, 29, 31].

In analyzing incentives in distributed systems, many papers model the node interaction as a one-shot game. However, many distributed settings involve repeated games in populations of interacting players where such approaches do not apply. In this paper, as our first contribution, we present a game-theoretic model of a distributed protocol that aims to capture some of the repeated aspects of such systems.

Furthermore, theoretical analyses often require high levels of abstraction to keep the models simple. This is required because involved models can become analytically intractable [19]. Thus, for modeling complex protocols, a theoretical analysis runs the risk of missing out on important variables that could have significant effects on protocol design. For instance, while designing a protocol, it is not uncommon for the designer to employ a variety of arbitrary design decisions and so-called “magic numbers”. Modifying any of these can have negative effects on the robustness of the protocol's incentives. In other words, there are many elements in complex protocols that could be gamed by strategic nodes.

A simulation based approach has been used by Axelrod [1] to model strategic interactions in repeated games. As our second contribution, taking inspiration from Axelrod, we aim to devise a method which can be used to analyze protocols more comprehensively. To that end, we present a simulation-based approach that we call *Design Space Analysis* (DSA). DSA combines the specification of a design space with an analysis of varying protocols within that space.

The specification of a design space comprises two steps: *Parameterization* and *Actualization*. *Parameterization* involves identifying salient design dimensions for the space, while *Actualization* involves specifying multiple implementations for the identified dimensions.

For an analysis of the design space, we present a solution concept, which we term the *Performance, Robustness*, and *Aggressiveness* (PRA) quantification. For a protocol Π, *Performance* is the overall performance of the system when all nodes execute Π (where performance is defined by the application); *Robustness* is the ability of a majority of the population executing Π to outperform a minority executing a protocol other than Π; and *Aggressiveness* is the ability of a minority of the population executing Π to outperform a majority executing a protocol other than Π. PRA quantification takes the form of a tournament in which each protocol competes against every other protocol. By evaluating each protocol in the space, the PRA quantification simulates strategic variants and predicts their effects.

We choose the domain of peer-to-peer (P2P) systems for applying and exploring our ideas, because there are many such deployed systems, for which incentive-compatible design is of primary importance to counter strategic behavior. We undertake the following steps. We apply a game-theoretic analysis to the popular P2P protocol, BitTorrent, and devise a more robust variant by incorporating the repeated aspects of the protocol (Section 2). Then we perform a Design Space Analysis of P2P file swarming systems. We run experiments on a cluster and discover that there are several protocols that do better than this variant with respect to Performance, Robustness, and Aggressiveness (Section 3 & 4). Finally, we implement modifications to BitTorrent and with experiments on a cluster, analyze some protocols discovered using DSA. We show that they yield higher system performance and robustness as compared to the reference implementation, thus demonstrating the effectiveness of DSA (Section 5).

2. GAME-THEORETIC ANALYSIS OF BITTORRENT

We consider one of the most popular P2P protocols, BitTorrent (BT), for our analysis. Our reason for choosing BitTorrent is that this protocol has probably been the most widely studied P2P protocol in the literature. A game-theoretic approach has often been applied to BitTorrent [12, 16, 26].

First, we present a model of BitTorrent as a *strategy* in a *game*. In game theory, a game is a description of a strategic interaction that includes the constraints on the actions that the players can take and the players' interests [22]. Then we present an analysis of this model for multiple bandwidth classes. Under our assumptions, which are different from previous work [26], we show that BitTorrent is not a Nash equilibrium. Finally, we design a modification to BitTorrent, which is a Nash equilibrium.

We assume the reader is familiar with certain game-theoretic constructs such as the Prisoner's Dilemma (PD), a game between two players in which it is the dominant strategy of both players to defect.

2.1 BitTorrent as a strategy in a game

We explain the basics of the BitTorrent protocol from an iterated games setting perspective. Each peer plays a number of games with other peers in a given time period, following a Tit-for-Tat (TFT) like strategy. TFT is the strategy using which a player *cooperates* on the first move and then simply mimics what the other player did in the last round. In BitTorrent a peer cooperates with (i.e., uploads to) a certain number of preferred (fastest uploading) partners while it defects in the rest of the games. These are the 'regular unchokes' in BT terminology. Additionally, a peer also starts new games with other peers in search of better partners. These are 'optimistic unchokes' in BT terminology. In these games, a peer always cooperates unconditionally for some iterations. We do not model the seeders in BitTorrent as these do not affect our subsequent Nash equilibrium analysis. This is because we assume, like Chow *et al*. [4], that seeders interact uniformly with all peers.

We now present an analysis of our model in a system containing two classes of peers: fast and slow. The game interaction in Figure 1(a) captures the dynamics between a fast peer and a slow peer, where f is the upload speed of a fast peer and s is the upload speed of a slow peer. This game represents a single round in an iterated scenario, where the 'shadow of the future' is large (i.e., the payoff of subsequent moves is important relative to the previous move) and peers can form sustained relationships. It can be seen that given the payoffs, the dominant strategy for fast peers is to always defect on the slow peers. This is because when a fast peer cooperates with a slow peer, there is an *opportunity cost* associated with it. Opportunity cost is an important concept in economics. It is the cost of an alternative that must be given up in order to pursue a certain action [8].

We note here that by incorporating the 'shadow of the future' and opportunity costs in our 'game' we try to model two key aspects of the BitTorrent protocol usually ignored in traditional game-theoretical analysis. These aspects are: (a) the repeated interactions between peers; and (b) the wide choice of partners that peers can have.

A fast peer's opportunity cost in cooperating with a slow peer is a missed interaction with another fast peer. When a fast peer cooperates with a slow peer, it gets a negative utility of $s-f$. It gets s from the slow peer but on the other hand, loses out on a potential f from a fast peer. Conversely, for the slow peers, the dominant strategy is to always cooperate with the fast peers. A slow peer on defecting against a fast peer gets f from the fast peer and can form a relationship with a slow peer, where it gets $s-f$ (where $-f$ is the opportunity cost of cooperating with a slow peer), thus getting a final utility of $f+(s-f)=s$. Figure 1(b) depicts this scenario: a slow peer responds (with cooperation in the form of a 'regular unchoke' slot) upon being optimistically unchoked by a fast peer, while the converse does not hold. In light of this, we note here that the Prisoner's Dilemma is not an accurate model for BitTorrent under heterogeneous classes of peers. Instead, the way BitTorrent implements the interaction of a slow peer with a fast peer, resembles an interaction in the *Dictator game*, a game in which one player proposes to do something, while the other has no choice but to respond passively without any strategic input into the decision. It also resembles a game which has been called by some as the *One-Sided Prisoner's Dilemma* [28]. For simplicity, we refer to it here as the *BitTorrent Dilemma*.

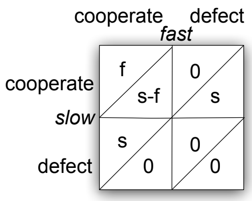
Next we give an analytical model of BitTorrent for multiple bandwidth classes, using the BitTorrent Dilemma game as depicted in Figure 1(a).

2.2 Analytical model of BitTorrent Dilemma

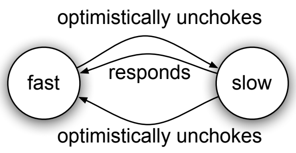
In this section, we model the BitTorrent Dilemma game with multiple bandwidth classes of peers. We seek to calculate the expected number of games that a peer c from a particular class, with payoffs defined according to Figure 1(a), can win against other peers, where winning means getting cooperation from others.

In the remainder of this section we derive the formulae for the expected number of games that peer c wins against other players from different classes. We note that there are two types of games that a player c can win: 1) the games that it wins when others *reciprocate* to it; and 2) when other players start a new game with c and in line with TFT, cooperate unconditionally, thereby giving c a *free game win*.

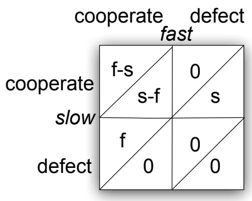
We use the notation summarized in Table 1. Note that we assume, for notational simplicity, that the number of new partners that a peer cooperates with unconditionally (number of optimistic unchoke slots in BT terminology) is equal to 1. Moreover, it is also



(a)



(b)



(c)

Figure 1: Analysis of the BitTorrent Dilemma: (a) The payoffs of the BT Dilemma for slow and fast peers; (b) An abstract illustration of interaction between slow and fast peers; (c) Modified BT payoffs in view of slow peers' opportunity costs.

assumed that there are always enough peers to exchange a particular piece of content.

Table 1: Model parameters. Classes are based on peers' bandwidth capacities.

Notation	Definition
N_A	number of TFT players in classes above c's class.
N_B	number of TFT players in classes below c's class.
N_C	number of TFT players in c's class.
U_r	number of players that c can reciprocate with simultaneously (number of regular unchoke slots in BT)
$E^r[X \to c]$	the expected number of games peer c wins against peers in the class $X \in \{A,B,C\}$, where A,B and C are the classes above, below and where peer c is from, respectively.
$E[X \to c]$	the expected number of 'free game wins' that peer c obtains from class $X \in \{A,B,C\}$.
N_r	$N_A + N_B + N_C - U_r - 1$

First, we calculate the expected number of games that can be won against higher classes. We assume that N_A is greater than U_r; thus, as per Figure 1(a), players employing TFT in higher classes will not reciprocate with peer c. Therefore,

$$E^r[A \to c] = 0.$$

However, as per the TFT policy, peers from higher classes, unknowingly, do offer first move cooperation to peers from lower classes in search for better partners. The probability that peers in class C are offered a 'free game win' by a peer from the higher classes is N_C/N_r, giving $E[A \to C] = N_A \times N_C/N_r$, which is the expected number of 'free game wins' that peers from higher classes offer to players in the considered class C. This leads to

$$E[A \to c] = N_A/N_r.$$

For the expected number of games won against the lower classes, using similar reasoning as above, we obtain

$$E[B \to c] = E^r[B \to c] = N_B/N_r.$$

The expected value for the number of games in which a peer c gets reciprocation from peers in the same class is U_r *minus* the number of 'free game wins' that peer c obtains from the higher classes (to which c always reciprocates as per its dominant strategy, thus breaking its relationship with a partner from the same class), *minus* the expected number of 'free game wins' that at least one of c's current partners gets from the higher classes (to which the partner reciprocates, thus breaking its relationship with c). This leads to

$$E^r[C \to c] = U_r - E[A \to c] - K, \tag{1}$$

where $K = 1 - \left((1 - E[A \to c])(1 - \frac{1}{U_r})\right)^{U_r}$. Finally, the number of peers in contention for 'free game wins' by peer c in the same class is $N_C - 1 - E^r[C \to c]$, which gives

$$E[C \to c] = (N_C - 1 - E^r[C \to c])/N_r.$$

2.3 Is BitTorrent TFT a Nash equilibrium?

Under certain assumptions, it was shown in [26] that the TFT strategy as implemented in BitTorrent is a Nash equilibrium. However, by modifying the abstraction to incorporate more detail in our model, and taking a cue from formula (1), we find that BT is not a Nash equilibrium. In our model, the payoff structure in BitTorrent can be modified to devise a new protocol. This protocol can individually, in a population of all BitTorrent peers, do better than the BitTorrent peers, thereby showing that BitTorrent is not a Nash equilibrium.

Next, we discuss how we devise a protocol improvement called *Birds*[1] by modifying the payoffs in the BitTorrent Dilemma.

Birds: Modifying BitTorrent's payoffs. A fast peer upon being optimistically unchoked by a slow peer does not reciprocate, as given by Figure 1(a), because a fast peer realizes the opportunity cost of reciprocating to a slow peer, which is a missed interaction with another fast peer. A slow peer reciprocates to a fast peer because in its view there is an opportunity cost in defecting against a fast peer, which is a missed chance to form a long-term relationship with a fast peer. However, as stated, given the scenario described in Figure 1(a), this does not happen. This suggests that the payoff as calculated by a slow peer in the BitTorrent Dilemma could be modified. There is no opportunity cost in defecting against a fast peer. In fact there *is* an opportunity cost in cooperating with it: missing out on a sustained relationship with another slow peer. Therefore, in order to account for this fact, we modify the payoff structure of the BitTorrent Dilemma according to Figure 1(c) so that the dominant strategy of both slow and fast peers is to defect against each other. This new payoff structure leads to a new protocol that we call Birds and reflects the behavior of peers following this protocol.

Analytically, this leads to

$$E^r_B[A \to c] = E^r_B[B \to c] = 0,$$

and obviously

$$E^r_B[C \to c] = U_r.$$

For the 'free game wins', there is no change as compared to BitTorrent. For the interactions of peers in the same class, we find that we can use the same argument as used for BitTorrent, thus

$$E_B[C \to c] = (N_C - 1 - U_r)/N_r.$$

[1]Birds of a feather stick together. Using this protocol peers try to stick with others from their own bandwidth class.

For a proof of Birds being a Nash equilibrium and BitTorrent not being a Nash equilibrium, we refer the reader to the Appendix.

A practical approach to deploy Birds. We provide a simple, practical approach to incorporate Birds in current BitTorrent clients. We propose to change the peer selection policy of BitTorrent, so that it reciprocates not to the fastest peers but to those peers that are closest to its own upload bandwidth. Given Figure 1(c), a peer in Birds needs to sort others in increasing order of their *distance* to its own upload bandwidth.

We define a peer's distance to another peer to be equal to the absolute difference between its upload speed and the upload speed of the other peer. We note that a similar approach has been used for replica placement in P2P Storage Systems [30].

2.4 Discussion

We have applied a game-theoretic analysis to the popular P2P protocol, BitTorrent. Considering BT as a strategy in iterated games, and using the concept of 'opportunity costs', we were able to unearth new protocol aspects. Using a different abstraction as compared to previous work [26] under which BT is a Nash equilibrium, we demonstrated that in our abstraction, BitTorrent is not Nash equilibrium. We also devised a protocol variant that is a Nash equilibrium under our assumptions.

We now need to consider what we gain from our equilibrium analysis and what exactly is meant when a protocol is said to be robust. It was only subsequent to the proof that BitTorrent is a Nash equilibrium [26], that questions about BitTorrent's robustness were raised. Locher *et al.* [17] showed that BitTorrent's TFT policy is vulnerable to attack from an *Always Defect* strategy. Later on, yet another BT exploit was devised based on an adaptive policy for number of partners and variable rate of reciprocation [24]. Even in our case, simply by choosing an abstraction that incorporates the interactions between various classes of peers in more detail, we showed that BT is not a Nash equilibrium.

The literature of game-theoretic modeling of distributed systems often relies on the analytical analysis of simple models [30, 26]. The results of such analyses, as we have seen, can sometimes be misleading. In many other areas of networking, simulations are used when a faithful model of the real system is far too complicated, and it is not clear which details one can omit without changing the nature of the results. In the next section, we present Design Space Analysis, an approach inspired by the work of Axelrod [1], which uses a simulation based methodology for modeling incentives in complex protocols.

3. DESIGN SPACE ANALYSIS

We wish to design distributed protocols that maximize performance of the system under the assumption that protocol variants may enter the system. We present Design Space Analysis (DSA), a simulation based method, which emphasizes the specification and analysis of a design space, rather than proposing a single protocol. First, we list the key elements of DSA. Then we present the *Performance, Robustness, Aggressiveness* (PRA) quantification, a solution concept within DSA.

3.1 Key elements of DSA

We consider the elements that are integral to Design Space Analysis.

Flexible behavioral assumptions. In DSA, we relax behavioral assumptions. Specifically, we do not limit ourselves to the rational framework, where nodes are supposed to be self-interested. By foregoing the assumptions entailed in this framework, we consider a great variety of protocols, which may not necessarily be rational. Protocols may, in the words of Axelrod [1], "simply reflect standard operating procedures, rules of thumb, instincts, habits, or imitation".

Specification of design space. In DSA, keeping in view that complex protocols have many elements that can be gamed by strategic nodes, a design space should encompass relevant details that can affect the incentive structure. Design space specification occurs at two levels: i) *Parameterization*, which involves determining the salient dimensions of the design space, and ii) *Actualization*, which involves specifying a host of actual values for every individual dimension.

The specification of the design space can be inspired by consulting the relevant literature and analyzing existing systems. As an example, the Parameterization phase of the design space for *Gossip Protocols* [13] could result in the following salient dimensions: i) Selection function for choosing partners for exchanging data, ii) Periodicity of data exchange iii) Filtering function for determining data to exchange, iv) Record maintenance policy in local database.

The Actualization of this example design space for gossip protocols could be: For Selection Function, following policies could be used : 1) *Random:* Choose partners randomly; 2) *Best:* Choose partners who have given the best service; 3) *Loyal:* Choose most loyal partners; 4) *Similarity:* Choose partners based on similarity; etc. Similarly, different values could be chosen for each of the other dimensions.

An example of specifying a design space, with both the parameterization and actualization phases, will be described in detail in Sections 4.1 and 4.2 when we apply DSA to P2P file swarming systems.

Systematic analysis of the design space. In DSA, a desired feature of all solution concepts is a systematic exploration of the design space. This exploration could either follow an exhaustive approach, e.g., a parameter sweep, or a heuristic based approach. By a thorough scan of the space, DSA solution concepts can anticipate strategic variants and predict their effects. Heuristic based approaches can provide partial solutions relatively fast, however, without any guarantees on the level of goodness of the measures.

3.2 The PRA quantification

We now present the PRA quantification, a solution concept within DSA. We note that other solution concepts within DSA could also be devised. Using PRA, we can characterize any protocol, from a given design space, over three measures (or dimensions). For a given protocol Π, these three particular measures, are:

- Performance - the overall performance of the system when all peers execute Π (where performance is determined by the application);
- Robustness - the ability of a majority of the population executing Π to outperform a minority executing a protocol other than Π;
- Aggressiveness - the ability of a minority of the population executing Π to outperform a majority executing a protocol other than Π.

We formulate a way to assign values to each of the three measures normalized into the range $[0, 1]$. Hence the properties of any given Π can be characterized as a point within a three-dimensional Performance, Robustness, Aggressiveness (PRA) space. The importance of these three measures is evident from the literature. Performance and Robustness have been studied extensively [2, 12, 24].

Table 2: Existing protocols/designs mapped to our generic P2P protocol design space.

Protocol	P2P Replica Storage [30]	GTG [21]	Maze [32]	Pulse [23]	BarterCast [20]	Private BT Communities
Peer Discovery	Gossip based	orthogonal	Central server	Gossip based	Gossip based	Central server
Stranger Policy	Defect if set of partners full	Unconditional cooperation	Initialized with points	Give positive score	Unconditional cooperation	Initial credit
Selection Function	Closest to own profile	Sort on Forwarding Rank	Ranked on points	Missing list, Forwarding list	Rank/Ban according to reputation	Credits or sharing ratio above certain level
Resource Allocation	Equal	Equal	Differentiated according to rank	Equal	orthogonal	Equal / Differentiated according to credits

Aggressiveness has not been explicitly studied but the numerous papers that present protocol variants [11, 15, 16] suggest the need for an aggressiveness measure to determine the effectiveness of a new protocol variant in a population of peers following some other protocol(s).

It is desirable, in open systems in which strategic variants can enter, to design protocols which maximize all three measures. However, it can be conjectured that there will often be a trade-off between them. For example, one may design protocols with high performance but low robustness or conversely high robustness and low performance.

We now define more precisely how we can map a given protocol Π, which can be expressed as a point in the design space, to a point in the PRA space; formally, we define a function $S : D \rightarrow [0,1]^3$, where D is the design space.

We assume that for each peer in a system of peers we can calculate a utility which quantifies individual performance. The measure of performance is application specific, such as download speed in P2P file swarming systems. Given this we define the performance P of protocol Π as the sum of all individual utilities in a population of peers executing Π normalized over the entire protocol design space. Hence, $P = 1$ indicates the best performance obtained from any protocol in the design space.

We define the Robustness R for protocol Π as the proportion of all other protocols from the design space that do not outperform Π in a *tournament*. A tournament consists of multiple *encounters* in which protocol Π plays against all other protocols in turn. An encounter is a mixed population of peers executing one of two protocols. The winning protocol is that which obtains the higher average utility for the peers executing it.

Aggressiveness A for protocol Π is defined in the same way as Robustness, but here Π is in the minority.

4. APPLYING DSA TO P2P FILE SWARMING SYSTEMS

In this section, we describe our methodology for applying DSA to P2P file swarming systems. First, we Parameterize a generic P2P design space. Next, based on this generic space, we Actualize a specific file swarming design space. Subsequently, we apply the PRA quantification on this space. Finally, we present the results of our analysis.

4.1 Parameterization of a Generic P2P Protocol Design Space

We have identified the following salient dimensions applicable to a large variety of P2P systems.

Peer Discovery: In order to perform productive peer interactions, it is necessary to find other partners. For example, when a peer is new in the system, looking for better matching partners or existing partners are unresponsive. The timing and nature of the peer discovery policy are the important aspects of this dimension.

Stranger Policy: When interacting with an unknown peer (stranger), past history cannot be used to inform actions. It is therefore necessary to apply a policy to deal with strangers. The way peers allocate resources to strangers is an important aspect of this dimension.

Selection Function: When a peer requires interaction with others this function determines which of the known peers should be selected. This could include, for example, past behavior (through direct experience or reputation system), service availability and liveness criteria.

Resource Allocation: During peer interactions resources must be allocated to the selected peers (given by the selection function). The way a peer divides its resources among the selected peers, defines the resource allocation policy.

Some existing implemented protocols and proposed designs are listed in Table 2. A protocol such as Give-to-Get (GTG) [21] employs unconditional cooperation with strangers as its *stranger policy*. A P2P replica storage design [30] presents a stranger policy which defects on strangers when its set of regular partners is full. For implementing the *selection function*, a deployed P2P reputation system like BarterCast [20] ranks peers in order of reputation score and also proposes to ban peers below a certain reputation score. P2P replica storage selects those peers which are closest to the selecting peer's own storage capacity. This selection policy is identical to the one proposed by us for implementing Birds in BitTorrent-like file-sharing systems. For *resource allocation*, Maze [32] allocates resources proportional to rank. These examples are a few implemented systems and proposed designs; a large variety of P2P systems that rely on eliciting cooperation from participating nodes rely on similar dimensions.

4.2 Actualization of a Specific P2P Protocol Design Space

We define some specific actualizations of a BitTorrent-like file swarming system, as described in Section 2, based on the general design space of Section 4.1. The ideas behind these actualizations have been taken directly from, or inspired by, various works on cooperation done in P2P and also some works done in biology and social sciences in general [1, 10, 25]. We were motivated to take inspiration from other fields, because eliciting cooperation in decentralized settings is a general problem that has been well-studied.

For the **Stranger Policy**, we define three different actualizations and a value h for the number of strangers to cooperate with:

B1) *Periodic:* Give resources to up to a certain number of strangers periodically.

B2) *When needed:* Only give resources to strangers when set of regular partners is not full. This particular implementation has been inspired by [11].

B3) *Defect:* Always defect on strangers, i.e., give nothing to strangers.

We set h, the number of strangers to cooperate with at any given time, to be in the range $[1,3]$. This gives $3 \times 3 = 9$ different stranger policies. We further add one more stranger policy, where the number of strangers is zero. This gives a total of 10 different stranger policies.

We sub-divide the **Selection Function** into three parts: a candidate list, a ranking function over that candidate list, and finally a value k for the number of peers to select from the ranked candidate list.

For the 'Candidate List' we define two actualizations:

C1) *TFT*, used by default in BitTorrent, using which a peer only places those peers in the candidate list who reciprocated to it in the last round.

C2) *TF2T*, using which a peer places those peers in the candidate list who reciprocated to it in either of the last two rounds. TF2T has been taken from [1].

For the 'Ranking Function', we define six different actualizations:

I1) *Sort Fastest*, ranks peers in order of fastest first.

I2) *Sort Slowest*, ranks peers in order of slowest first.

I3) *Sort Based on Proximity*, ranks peers in order of proximity to one's own upload bandwidth, as in Birds.

I4) *Sort Adaptive*, ranks peers in order of proximity to an *aspiration level*, which is adaptive and changes based on a peer's evaluation of its performance. This has been inspired from [25].

I5) *Sort Loyal*, ranks peers in order of those who have cooperated with the peer for the longest durations. This has been inspired by [10].

I6) *Random*, does not rank peers and chooses them randomly. This has been inspired by [15].

After applying the ranking function, a peer chooses the k top peers, where k is the maximum number of partners that a peer can have. Currently, we set k to be in the range $[1,9]$. This results in $2 \times 6 \times 9 = 108$ different possibilities for the selection function. We further add one more protocol, where the number of selected peers is zero. This gives a total of 109 different selection policies.

For **Resource Allocation**, we define three actualizations:

R1) *Equal Split*, gives all selected peers equal resources (upload bandwidth).

R2) *Prop Share*, gives others proportional to what they gave in the past. This has been inspired by [16].

R3) *Freeride*, gives nothing to partners.

Based on the above, the total number of unique protocols comes to $10 \times 109 \times 3 = 3270$. We note that this number can be larger or smaller based on what, and how many, specific implementations in the space, designers want to explore[2]. Our purpose here is to show the practicality of the DSA analysis by analyzing a considerable space of unique protocols.

[2]For instance, we do not consider variants for resource allocation to strangers. Also, we do not consider Peer Discovery.

4.3 Conducting the PRA quantification

First we describe our simulation model. Then we discuss our methodology for conducting the PRA quantification on the design space described in Section 4.2.

4.3.1 Simulation Model

We use a cycle-based simulation model, in which time consists of *rounds*. For *peer discovery* we assume that all peers can connect to each other. In each round, a peer decides to upload to a given number of peers based on some *selection* criterion. It uses its *resource allocation* policy to decide how much to give to each of the selected partners. Furthermore, it decides to cooperate with strangers based on its *stranger policy*. A peer also maintains a short history of actions by others. At the same time a peer also has some rate of requesting services from other peers that depends on specific actualizations. This is the basic model on top of which we explore the design space of Section 4.2. We run our simulation experiments with 50 peers, which is a good approximation of an average BitTorrent swarm-size [9]. These peers interact with each other for 500 rounds. We use a cluster for running our experiments. In order to lend realism to our experiments, we initialize the peers using the bandwidth distribution provided by Piatek *et al.* [24]. We assume that all peers always have data that others are interested in.

4.3.2 Methodology

Based on the PRA quantification, as described in Section 3.2, we first measure the Performance of each protocol in the space. For each protocol Π, we run simulations in which all peers execute Π and measure the average performance of the population. We perform 100 runs for each protocol. In these experiments we define average performance as throughput of the population.

Next we run Robustness experiments. We run simulations, where each protocol plays against every other protocol. We refer to a competition in which two protocols are pitted against each other as an *encounter*. For each encounter, the peer population is split up into two equal halves where half the peers execute Π and the other half executes another protocol. We chose 50% because this is the highest number that an invading protocol can have. Anything higher than 50% means that the invading protocol actually becomes the majority protocol. We hypothesize that if a protocol is robust when 50% of the population executes another protocol, then it will be robust against small invading populations. To verify this hypothesis, we also conduct simulations with the population split up into 90-10, where 90% of the peers follow protocol Π, while 10% execute other protocols in the space, and observe similar results (the Pearson's correlation coefficient of the two sets of results is 0.97). We do 10 runs for each particular encounter between two protocols. This means that a protocol Π plays against the same protocol ten times. For each run, we compare the average performance of Π with the average performance of the other protocol. If the performance of Π is greater than the performance of the other protocol, we mark it as a *Win* for Π, otherwise we mark it as a *Loss* for Π. The robustness value for Π is calculated by number of games that it wins against all opponents in all runs divided by the total number of games that it plays, which is constant for all protocols.

For Aggressiveness we use the same setup as for Robustness experiments, with the difference that the population is so divided that 10% of the peers execute Π while the rest execute another protocol.

Next, we present results of applying the PRA quantification over the design space[3] described in Section 4.2.

[3]We note here that the entire series of simulations for PRA took around 25 hours on a 50-node dual-core cluster.

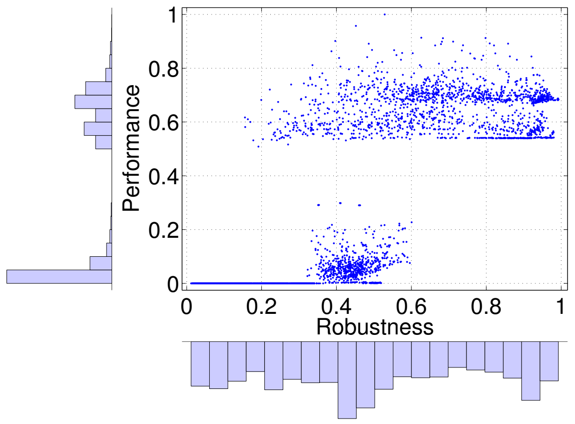


Figure 2: Scatter plot of all 3270 protocols in the design space with Robustness against Performance. The results presented here are a synthesis of over 107 million individual simulation runs. Histograms are also shown.

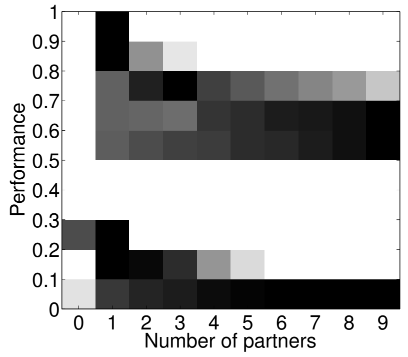


Figure 3: Normalized Performance histograms for different partner values. Darker squares represent high 'partner value' frequency for a particular Performance interval.

4.4 Results and Discussion

Figure 2 shows all the 3270 protocols actualized in Section 4.2, with their normalized Robustness and Performance values. Given the methodology of conducting the PRA quantification as described in Section 4.3.2, this figure represents a synthesis of 107 million individual runs. For performance, each point represents the average normalized performance of a protocol Π, over 100 runs. The robustness values are calculated as described in Section 4.3. The maximum variance in the runs for each protocol's performance and robustness was as low as 0.0014 and 0.0206, respectively.

Performance. We shall start by looking at different regions of Figure 2. It can be seen that numerous protocols have both very low performance and robustness in the range $[0, 0.2]$ (as can be seen from the large histogram bars in the bottom left hand corner). Upon inspection we discover that most of them are freeriders, although different kinds of freeriders. The freeriders with low robustness usually defect on strangers, while freeriders with low performance usually defect on their partners. The maximum performance that such freeriders get is 0.31. This can be seen in the form of two clusters in Figure 2, where the lower and upper clusters are below and above 0.4 performance, respectively.

The lower cluster for performance contains all freeriders that do not reciprocate with their partners. It also contains some freeriders that defect on strangers. The upper cluster for performance does not contain any freeriders that defect on their partners but interestingly does contain freeriders who defect on strangers. In fact, the protocol with the highest performance (of 1), is the one that always defects on strangers, has the *Sort Slowest* ranking function, and maintains one partner. We try to dissect why this particular protocol performs so highly. By defecting on a stranger, a protocol *in effect* uploads nothing to the stranger. Since, all peers follow the *Sort Slowest* ranking function, a peer p_1 on downloading nothing from another peer p_2, immediately discards its partner p_3 and chooses p_2 to be its partner. Peer p_3 now finds itself partner-less. However, it can also quickly find a partner for itself by uploading nothing to another peer. Thus by following this protocol, peers rarely find themselves without a fully occupied partner set and can always download at full speed.

Thus, counter-intuitively, by maintaining a single partner, by not uploading anything to strangers and by employing the *Sort Slowest* ranking function, a very high performing protocol can be designed. It is imperative of course in this case that the *resource allocation* method should *not* be *Prop Share*. This is because *Prop Share* will ensure that a peer on getting nothing from another peer, also uploads nothing in response. If that happens, the entire population that follows this protocol will fail to bootstrap.

In Figure 3 we observe that in fact many of the top performing protocols are those with one partner. In Figure 3, for each performance interval, darker squares represent higher relative frequency of the number of partners. The empty white spaces in Figure 3 reflect the empty spaces in the scatter plot and the histogram of the Performance values in Figure 2. All the top 15 performing protocols maintain one partner each and the overall trend in the top performing protocols shows a low number of partners. In the top hundred performing protocols only 11 maintain more than 2 partners. It can be seen from Figure 3, starting from the top, till the performance interval $[0.7, 0.8]$, the frequency of low number of partners is relatively higher than the frequency of high number of partners.

We analyze these high performing protocols with a low number of partners more closely now. We have already discussed the features of the highest performing protocol, which uses the *Sort Slowest* ranking function. However, not all high performing protocols with low number of partners have the same features. Many of them employ the *Sort Loyal* ranking function. With the *Sort Loyal* ranking function, it can be conjectured that peers which come to form cooperative relationships early on, stay committed in their relationships. This stability of relationships increases the performance of the system, because at no time do peers find themselves short of partners.

Apart from this, many such protocols often also use the *When needed* stranger policy. This policy also leads to more committed partnerships. With the *When needed* stranger policy, peers only cooperate with strangers when their set of partners is not full. Thus once its partner set is full, a peer will not cooperate with strangers. By not cooperating regularly with strangers, a peer protects itself from breaking relationships by avoiding temptation. This is because when a peer p defects against a stranger, it does not get back anything in response from the stranger. In this way peer p does not discover how much better or worse the stranger is than its current partners, and thus continues to stick with them.

It could be assumed that in the presence of churn, protocols with low number of partners will not perform so well. However, we ran Performance tests for the whole space under churn rates of 0.01 and 0.1 per round, and found that it was still the protocols that employed

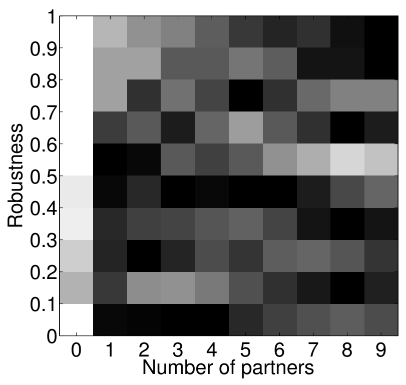


Figure 4: Normalized Robustness histograms for different partner values. Darker squares represent high 'partner value' frequency for a particular Robustness interval.

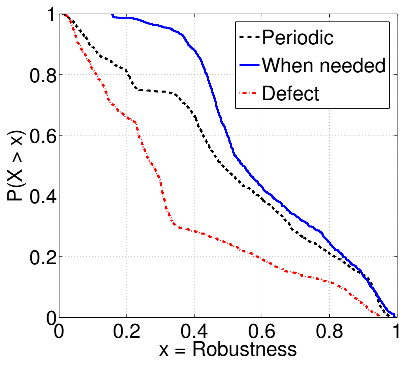


Figure 5: Complementary CDF plots of Robustness of different stranger policies.

a low number of partners that performed the best. Thus, it can be claimed that having low number of partners is a desirable feature of high performing protocols, given that everyone in the population runs the same protocol.

Robustness. It is interesting to compare Figure 3 with Figure 4. While a low number of partners seems to be a good choice for high performance, the situation is reversed when it comes to robustness. In Figure 4, we observe that most of the highly robust protocols have a high number of partners. This can be seen in the top right hand corner of the figure. This is intuitive, because protocols that employ a low number of partners would very likely perform worse in face of an invading protocol that employs high number of partners. This is because, in case of an invasion, peers employing a protocol that maintains a high number of partners are less likely to find themselves short of partners as compared to peers who follow a protocol that maintains a low number of partners. Hence, in the case that some partners defect, peers with a high number of partners can continue to download at high speeds while peers with a low number of partners are likely to suffer with poor speeds. To put it straightforwardly, a peer that maintains 9 partners will suffer less when one of its partners defects on it, as compared to a peer that only maintains one partner.

As we are considering the robustness of protocols, it can be seen from Figure 2 that there are some protocols that are highly robust with robustness values above 0.99. By analyzing these highly robust protocols, we discovered their interesting properties. Including

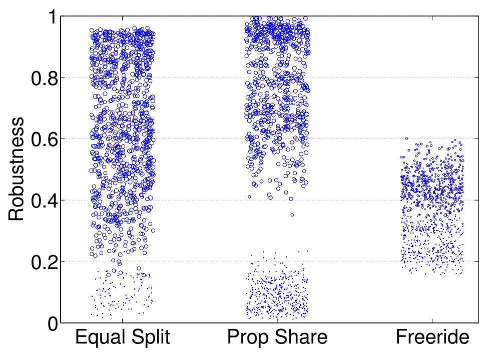


Figure 6: Robustness values using different resource allocation methods. Each circle is a unique protocol. Bigger circles represent better performance.

the most robust protocol in that cluster, these protocols use a combination of the *When needed* stranger policy, the *Sort Fastest* ranking function and the *Prop Share* reciprocation function. Figure 5 shows that only those protocols which use the *When needed* stranger policy reach robustness levels greater than 0.99. Similarly, it can be seen from Figure 7 that protocols which use the *Sort Fastest* ranking function, are the most robust protocols. Figure 6, shows that even though the *Equal Split* resource allocation policy does quite well, it is the *Prop Share* policy that manages to get robustness values greater than 0.99.

Since this is a vital point, as it tells us about the features of protocols that are almost completely robust, we consider these properties in detail. As discussed previously, the *When needed* stranger policy only cooperates with strangers when its set of partners is not full. The *Sort Fastest* ranking function, as the name implies, ranks peers in decreasing order of their speed, and finally the *Prop Share* resource allocation policy, allocates resources to peers in proportion to their contribution.

This combination, first of all, validates the claims about the robustness of the *Prop Share* mechanism [16]. However, it should be noted that, unlike their proposal, these protocols do not reciprocate with everyone. In fact, the most robust protocol maintains only 7 partners. Secondly, the combination of *When needed* with *Prop Share* is very important to note. In [16] a cryptographic bootstrapping technique was proposed to avoid exploitation by freeriding strangers. Our results suggest that the combination of the *When needed* stranger policy with the *Prop Share* resource allocation policy is a practical and lightweight alternative for designing robust protocols.

Trade-off between Performance and Robustness. Looking at the very robust protocols, we note that most of them are not among the high performing protocols. This suggests a trade-off between performance and robustness.

However, looking at the top right hand corner of Figure 2, we can see that there are at least some protocols that are robust and also have high performance (with robustness and performance values above 0.8). On inspection we find that there are 9 such protocols and *all* of these protocols follow the *Sort Loyal* ranking function. No other dimension (such as resource allocation, stranger policy, etc.) is uniform across all 9 protocols. *Sort Loyal* cooperates with those other peers who cooperate with it for the longest durations. It could have been assumed that a ranking policy like *Sort Loyal* would not fare very high in terms of robustness. This is because of the danger that a fast peer that employs the *Sort Loyal* ranking

Table 3: Multiple linear regression analysis applied for the PRA measures of the whole search space. The adjusted R^2 values are reported in the second row. The standard errors for all the variables are less than 0.012. Significance level is indicated as 'OK' if it was less than 0.001.

	Performance			Robustness			Aggressiveness		
	(adj.$R^2 = 0.68$)			(adj.$R^2 = 0.52$)			(adj.$R^2 = 0.61$)		
variable	estimate	t value	sign.	estimate	t value	sign.	estimate	t value	sign.
(intercept)	0.661	64.372	OK	0.813	73.286	OK	0.801	96.300	OK
$\log(\tilde{k})$	−0.008	−2.487	–	0.035	10.334	OK	0.037	14.591	OK
$\log(\tilde{h})$	0.109	33.121	OK	0.115	32.287	OK	0.092	34.340	OK
B2	0.008	1.046	–	0.026	3.010	OK	0.026	4.012	OK
B3	−0.206	−26.257	OK	−0.241	−28.399	OK	−0.190	−29.778	OK
C2	−0.066	−10.567	OK	−0.045	−6.576	OK	−0.039	−7.716	OK
I2	0.023	2.151	–	−0.214	−18.186	OK	−0.212	−24.000	OK
I3	0.020	1.831	–	−0.199	−16.911	OK	−0.155	−17.503	OK
I4	0.022	1.975	–	−0.230	−19.517	OK	−0.193	−21.796	OK
I5	0.031	2.843	OK	−0.074	−6.262	OK	−0.097	−11.004	OK
I6	−0.009	−0.796	–	−0.082	−7.080	OK	−0.090	−10.259	OK
R2	−0.194	−25.260	OK	−0.093	−11.188	OK	−0.053	−8.511	OK
R3	−0.544	−70.848	OK	−0.220	−26.562	OK	−0.253	−40.697	OK

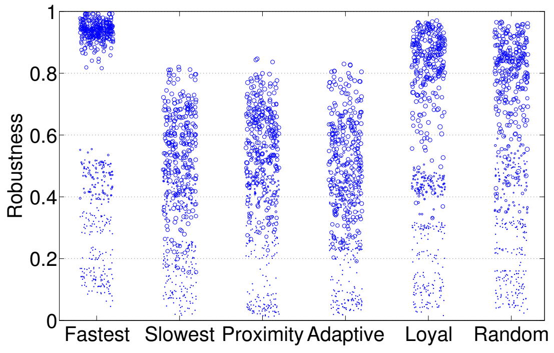


Figure 7: Robustness values using different ranking functions. Each circle is a unique protocol. Bigger circles represent better performance.

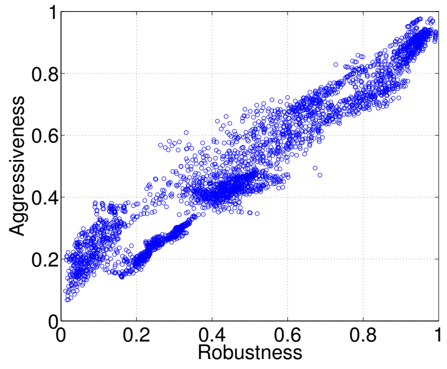


Figure 8: Scatter plot of robustness and aggressiveness values of the protocols. The Pearson's correlation coefficient is 0.96.

function, could get stuck with very slow peers (who follow another protocol) that keep cooperating with it. However, it is interesting to note that the highest robustness achieved by a protocol that sorts others based on loyalty, is actually a very high 0.97.

Aggressiveness. In Figure 8 we see that Robustness and Aggressiveness are linearly correlated with a Pearson's correlation coefficient of 0.96. This suggests that robust protocols are also very aggressive and there does not seem to be a major payoff between robustness and aggressiveness. We can conclude that the results for Robustness also hold for Aggressiveness.

4.4.1 Regression Analysis

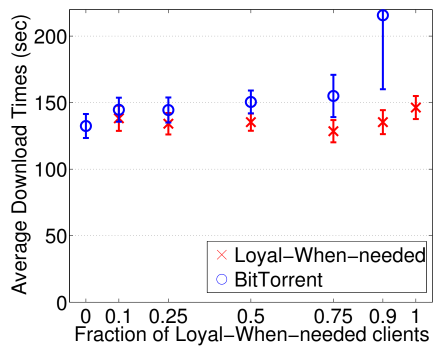
We applied multiple linear regression analysis for the whole protocol design space, which is reported in Table 3. Values of h and k (i.e. number of strangers and partners) were treated as numerical values (in the table $\tilde{h}$ and $\tilde{k}$ are the standardized values of h and k, respectively), whereas the other variables are categorical, thus were substituted by dummy variables.

These results serve as a summary of our previous results and also examine some dimensions which were not covered in the previous section. From Table 3 we can conclude the following: i) Choosing *Freeride* as a resource allocation policy (R3) has the biggest negative impact on Performance and Aggressiveness and it is also has a big negative impact on Robustness. ii) Another bad choice is the *Defect* stranger policy (B3). This policy has the biggest negative effect on Robustness. On the other hand, the *When needed* policy (B2) increases Robustness and Aggressiveness, but is not statistically significant for Performance. iii) Increasing the number of strangers to cooperate with increases all three, Performance, Robustness, and Aggressiveness values and this variable has the biggest positive effect on all three measures. iv) Higher number of partners results in an increase in Robustness and Aggressiveness, but not for Performance. v) We can see that *TF2T* strategy (C2) plays a consistent negative role for all three measures. vi) The choice of the ranking function has a big effect on Robustness and Aggressiveness. This is in line with Figure 7. However, choice of ranking function does not have significant impact on Performance, except for the *Sort Loyal* ranking function (I5), which increases Performance.

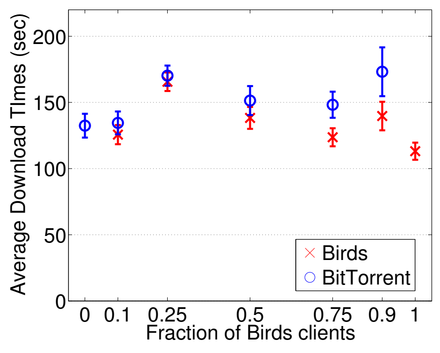
4.4.2 Birds according to Design Space Analysis

We devised a robust variant of BitTorrent in Section 2.3 using a game-theoretic analysis. Subsequently, we augmented game-theoretic analysis for protocol design with Design Space Analysis. We would now like to inspect if the results that we obtained from our game-theoretic analysis have held. How does Birds fare in the larger design space, under a more comprehensive and more realistic analysis?

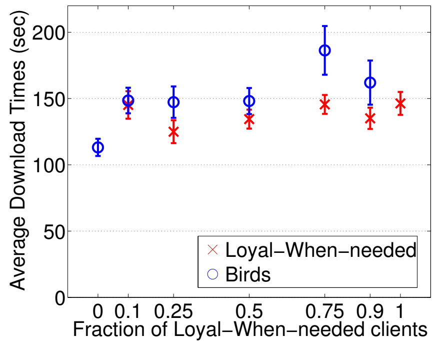
For the Performance measure, the best Birds variant, i.e., a protocol that at the very least ranks other by *Proximity* and employs



(a)



(b)



(c)

Figure 9: Encounters between three selected protocols.

Equal Split reciprocation, does well with a Performance value of 0.83 to rank at 30 among all 3270 protocols. In Robustness, Birds achieves a highest value of 0.76 and ranks at 714. For Aggressiveness, Birds achieves a highest value of 0.74 to rank at 630 among all protocols.

4.4.3 Discussion

Using DSA we were able to discover protocol variants that do well by employing interesting and counter-intuitive combinations of actualized dimensions. Some combinations lead to extremely high performance, while some lead to very high robustness. We also discovered a few protocols that are both highly robust and also have high performance.

The highly robust protocols are the best candidates for usage in open distributed systems, in which protocol variants may enter; whereas the protocols that achieve very high performance are perhaps more suited to closed systems, where incentives are not required. With the regression analysis on the whole design space we were able to measure the relative impacts of the different dimensions. This also gives new insights into practical protocol designs, and indicates the actualized dimensions that should be preferred and those that should be avoided.

Finally, we observed that Birds ranked very well in Performance and it is within the top quarter in Robustness. Given the fact that Birds was devised using a highly abstracted game-theoretic analysis, we claim to have: a) shown that a game-theoretic analysis is a useful tool which can be used to devise protocols through simple abstractions, that do reasonably well; and b) demonstrated the utility of DSA, by discovering *several* protocols that do better than Birds, in terms of Robustness and Performance.

5. VALIDATION OF DSA RESULTS

We discovered several interesting protocols using Design Space Analysis. In this section we want to explore how well DSA can be used to design deployable protocols. In order to prove the feasibility of using DSA to produce robust and high performing protocols that can be deployed, we modified an instrumented BitTorrent client provided by [14]. From the discovered protocols we choose one that uses the *Sort Loyal* ranking function and the *When needed* stranger policy because this variant resulted in both high Performance and Robustness according to DSA. We term this protocol as 'Loyal-When-needed'. Another variant was suggested by our Nash equilibrium analysis, which is the Birds protocol. Birds uses the *Proximity* ranking function. The third protocol type is the standard BitTorrent, which represents the baseline.

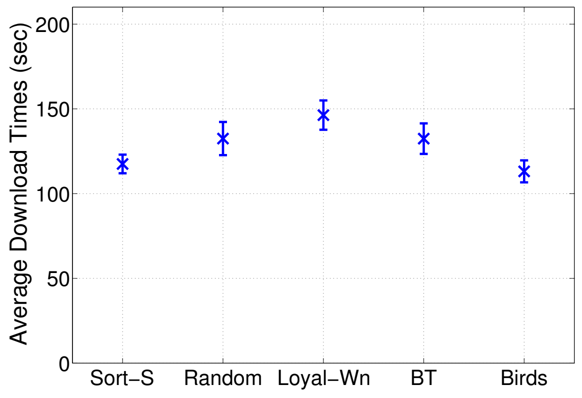


Figure 10: Performance of various protocols.

Experimental setup. In our experiments we pitted one protocol against another, with varying proportions of the two protocols. We adopted an experimental setup similar to [16] and [24]. The total number of leechers is 50. We setup one seeder with an upload bandwidth of 128 KBps. We setup a local tracker, with peers downloading 5MB files. We used the bandwidth distribution provided by [24]. Peers leave upon completing their download. The results of these runs can be seen in Figures 9 and 10, where each point in the figures represents the average over at least *10* runs and error bars mark 95% confidence intervals. Finally, we base our decision to use a cluster on the arguments and results in [27].

BitTorrent vs Loyal-When-needed. Figure 9(a) represents competitive encounters between BitTorrent and Loyal-When-needed clients. We can see the consistent trend of the Loyal-When-needed clients regarding the average download times: it is largely independent of the composition of the swarm. Moreover, Loyal-When-needed clients never do worse than BitTorrent, and they do significantly better if they are in a majority in the swarm.

Birds vs BitTorrent. Figure 9(b) represents a validation of our game-theoretic analysis from Section 2.3: Birds does as well or better than BitTorrent in all proportions. The difference is statistically significant if the proportion of Birds clients is more than or equal to three quarters. We can also conclude that the swarm with only Birds clients results in significantly better average download time than a swarm with only BitTorrent clients.

Birds vs Loyal-When-needed. Finally, in Figure 9(c) we compare the competitive encounters of Birds and Loyal-When-needed. We can immediately see that a swarm with only Birds clients results in better average download time. Together with the previous results, this validates our analysis using DSA, according to which Birds ranks high in performance (as discussed in Section 4.4.2). In line with our DSA results, the Loyal-When-needed protocol is more robust than Birds. The degradation in Birds performance becomes statistically significant when the two protocols compete; this is more evident when the Loyal-When-needed clients are in the majority.

Performance of various protocols. We now compare the performance of different protocols when all peers in the population execute the same protocol. For this, we consider two additional protocols that we discovered through DSA. Figure 10 shows that a protocol that we term Sort-S, together with Birds, fares the best when all peers in the swarm follow the same protocol. Sort-S is the interesting protocol that we discovered in our DSA analysis. It uses the *Sort Slowest* ranking function, defects on strangers and only has one partner. It is interesting to observe that a protocol that uses the *Sort Random* ranking function performs as well as BitTorrent. This recalls the results presented in [15]. We note that Figure 10 does not say anything about the Robustness of these protocols.

6. RELATED WORK

The study of incentive mechanisms for distributed systems has been a well-studied topic in the research community. Such works can be roughly categorized into the works, which came before the seminal papers by Feigenbaum and Shenker [6], and Dash *et al.* [5], and those that came subsequently. We concentrate on the latter works. Mahajan *et al.* [18] described their unsuccessful efforts at applying game theory for system design. They suggested that relaxing the notion of perfect selfishness could increase the applicability of game theory. We demonstrated that game theoretic analysis despite high level of abstraction, can result in a fruitful analysis, which, however, needs to be augmented with a detailed analysis.

Feldman *et al.* [7] applied an evolutionary game-theoretic analysis on a P2P design space. Their analysis differed from ours as it focussed on simple cooperation, defection and reciprocation, whereas we analyze a considerably vast space of protocols. Also, we apply the results of our analysis to BitTorrent to validate our approach.

Qiu *et al.* [26] under certain assumptions showed that BitTorrent is a Nash Equilibrium. We used a different abstraction and proved otherwise. Similarly Levin *et al.* [16] argued that BitTorrent is an auction, and with their model gained insights to develop a more robust variant. We showed that considering BitTorrent as a strategy in a game, also leads to insights, and we also developed a more robust variant.

Finally, many papers propose protocol variants not based on game-theoretic analysis [2, 15], but to our knowledge, ours is the first attempt to provide a comprehensive, simulation based approach.

7. CONCLUSION AND FUTURE WORK

We have presented Design Space Analysis (DSA), a simulation based approach that complements game-theoretic analysis of incentives in distributed protocols. DSA emphasizes the specification and analysis of a design space, rather than proposing a single protocol. We have shown that DSA can be used to gain an in-depth analysis of the properties of protocol variants, and it can be used for designing deployable protocols. For future work, we would like to explore if a solution concept similar to PRA quantification could be developed which explores the design space using a heuristic based approach. This could be needed in situations where a thorough scan of the design space becomes infeasible due to its size. We would also like to test DSA on distributed domains other than P2P. In conclusion, we note that DSA is a general method, which is practical and easy to apply, and we have demonstrated its merits by applying it successfully to P2P file swarming systems.

Acknowledgement

This work was partially supported by the Higher Education Commission (HEC) of Pakistan; and by the European Community 7th Framework Program through the QLectives and P2P-Next projects (grant no. 231200, 216217). We thank Márk Jelasity and his group at Szeged, Hungary for their valuable feedback during the early phase of this work. We would also like to thank the anonymous reviewers who have helped to improve the paper. Finally, we thank our shepherd, Scott Shenker for guiding us in the final stages of this work.

8. REFERENCES

[1] R. Axelrod. *The Evolution of Cooperation*. Basic Books, New York, 1984.

[2] A. Bharambe, C. Herley, and V. Padmanabhan. Analyzing and improving a BitTorrent network's performance mechanisms. In *INFOCOM*, 2006.

[3] C. Buragohain, D. Agrawal, and S. Suri. A game theoretic framework for incentives in P2P systems. In *IEEE P2P*, 2003.

[4] A.L.H. Chow, L. Golubchik, V. Misra. BitTorrent: An extensible heterogeneous model. In *INFOCOM*, 2009.

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

[6] J. Feigenbaum and S. Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *ACM DIALM*, 2002.

[7] M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *ACM EC*, pp. 102–111, 2004.

[8] R. Hahnel and A. Library. *The ABCs of political economy: A modern approach*. Pluto Press, 2002.

[9] T. Hoßfeld, F. Lehrieder, D. Hock, S. Oechsner, Z. Despotovic, W. Kellerer, and M. Michel. Characterization of BitTorrent swarms and their distribution in the Internet. *Computer Networks*, 55:1197–1215, 2011.

[10] D. Hruschka and J. Henrich. Friendship, cliquishness, and the emergence of cooperation. *Journal of Theoretical Biology*, 239:1–15, 2006.

[11] R. Izhak-Ratzin. Collaboration in BitTorrent systems. In *IFIP-TC Networking*, pp. 338–351, 2009.

[12] S. Jun and M. Ahamad. Incentives in BitTorrent induce free riding. In *P2PECON*, 2005.

[13] A.-M. Kermarrec and M. van Steen, editors. *ACM SIGOPS Operating Systems Review 41, Special Issue on Gossip-Based Networking*. 2007.

[14] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in bittorrent systems. In *ACM SIGMETRICS*, pp. 301–312, 2007.

[15] B. Leong, Y. Wang, S. Wen, C. Carbunaru, Y. Teo, C. Chang, and T. Ho. Improving peer-to-peer file distribution: winner doesn't have to take all. In *ACM APSys*, pp. 55–60, 2010.

[16] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent is an auction: analyzing and improving BitTorrent's incentives. In *ACM SIGCOMM*, 2008.
[17] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free riding in BitTorrent is cheap. In *HotNets-V*, 2006.
[18] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Experiences applying game theory to system design. In *ACM PINS*, pp. 183–190, 2004.
[19] G. Mailath. Do people play Nash equilibrium? Lessons from evolutionary game theory. *Journal of Economic Literature*, 36:1347–1374, 1998.
[20] M. Meulpolder, J. Pouwelse, D. Epema, and H. Sips. BarterCast: A practical approach to prevent lazy freeriding in P2P networks. In *IEEE IPDPS*, 2009.
[21] J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, and H. Sips. Give-to-get: Free-riding-resilient video-on-demand in p2p systems. In *SPIE/ACM MMCN*, 2008.
[22] M. Osborne and A. Rubinstein. *A course in game theory*. The MIT press, 1994.
[23] F. Pianese, J. Keller, and E. Biersack. PULSE, a flexible P2P live streaming system. In *INFOCOM*, 2006.
[24] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent. In *NSDI*, 2007.
[25] M. Posch. Win-Stay, Lose-Shift Strategies for Repeated Games–Memory Length, Aspiration Levels and Noise. *Journal of Theoretical Biology*, 198:183–195, 1999.
[26] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. *ACM SIGCOMM Computer Communication Review*, 34:367–378, 2004.
[27] A. Rao, A. Legout, and W. Dabbous. Can Realistic BitTorrent Experiments Be Performed on Clusters? In *IEEE P2P*, 2010.
[28] E. Rasmusen. *Games and information: An introduction to game theory*. Wiley-Blackwell, 2007.
[29] T. Roughgarden and É. Tardos. How bad is selfish routing. *Journal of the ACM*, 49:236–259, 2002.
[30] K. Rzadca, A. Datta, and S. Buchegger. Replica placement in p2p storage: Complexity and game theoretic analyses. In *ICDCS*, pp. 599–609, 2010.
[31] S. Shenker. Making greed work in networks: A game-theoretic analysis of switch service disciplines. *IEEE/ACM Transactions on Networking*, 3:819–831, 2002.
[32] M. Yang, Z. Zhang, X. Li, and Y. Dai. An empirical study of free-riding behavior in the Maze P2P file-sharing system. *Peer-to-Peer Systems IV*, pp. 182–192, 2005.

Appendix: BitTorrent Nash Equilibrium

In the following we use the notation introduced in Table 1 and the results from Sections 2.2 and 2.3.

In order to show that *BitTorrent is not a Nash equilibrium* (NE), we consider a swarm with $N-1$ BitTorrent (BT) peers and assume that one peer using the Birds protocol enters this swarm.

In this setup the expected number of games won by the BT clients against higher and lower classes do not change. On the other hand, for the Birds client only the formula for the expected number of games won against the peers from the lower classes changes to $E_B^r[B \to c]' = N_B/N_r$, which is the same as for the BT clients.

Now, we consider the class C where the peer using the Birds protocol is located. The expected values of the number of games that the peers win due to reciprocation from other peers in this class will be $E_B^r[C-c]' = U_r - K$ for Birds and

$$\begin{aligned} E^r[C\to c]' &= \frac{N_{C'}-U_r}{N_{C'}}(U_r-K-E[A\to c]) \\ &\quad + \frac{U_r}{N_{C'}}(U_r-E[A\to c]-K') \\ &= U_r-K-E[A\to c]-\frac{U_r}{N_{C'}}(K+K') \end{aligned}$$

for the BT clients, where $N_{C'} = N_C - 1$, and $K' = 1 - \left((1-E[A\to c])(1-\frac{1}{U_r})\right)^{U_r-1}$, which leads us to the fact that

$$E_B[C\to c]' > E[C\to c]'.$$

Regarding the 'free game wins', the formulae change to

$$E_B[C\to c]' = \frac{N_{C'}}{N_C}(N_C - E^r[C\to c]')/N_r,$$

$$E[C\to c]' = E_B[C\to c]' + \frac{N_C - E_B^r[C\to c]'}{N_C N_r},$$

which says that $E[C\to c]' > E_B[C\to c]'$; however,

$$E_B^r[C\to c]' + E_B^r[C\to c]' > E^r[C\to c]' + E[C\to c]'$$

holds. Thus, the peer using the Birds protocol, on average, wins more games than any of the BT clients, proving that BT is not a NE.

Now we show that *it is a NE when all peers in the swarm follow the Birds protocol*.

We assume that there are $N-1$ peers following the Birds protocol and one peer using the BT protocol enters this swarm. We give a formal proof for the case when this new peer uses BT; the other three cases (regarding class-based reciprocation) can be proved in the similar way.

First, we consider the games where peers get reciprocation. Neither the Birds peers nor the BT peer get anything from the higher and lower classes. For that particular class C, where the BT peer is located we have

$$\begin{aligned} E_B^r[C\to c]'' &= \frac{N_{C'}-U_r}{N_{C'}}U_r + \frac{U_r}{N_{C'}}(U_r - E[A\to c]) \\ &= U_r - \frac{U_r}{N_{C'}}E[A\to c], \end{aligned}$$

where $N_{C'}$ is the number of Birds in class C, i.e. $N_{C'} = N_C - 1$. Moreover, we have $E^r[C\to c]'' = U_r - E[A\to c]$; from here it is easy to see that $E_B^r[C\to c]'' > E^r[C\to c]''$.

'Free game wins' remain the same. The expressions for the same class become

$$E[C\to c]'' = \frac{N_{C'}}{N_C} \times \frac{N_{C'} - E_B^r[C\to c]}{N-U_r-1},$$

and

$$E_B[C\to c]'' = E[C\to c]'' + \frac{N_{C'} - E^r[C\to c]}{N_{C'}(N-U_r-1)},$$

Thus we conclude that $E_B[C\to c]'' > E[C\to c]''$ which completes our proof that Birds is a NE.