CS269I: Incentives in Computer Science
Lecture #5: Incentives in Peer-to-Peer Networks*

Tim Roughgarden†

October 10, 2016

1 Peer-to-Peer Networks: Some History

Ten years ago, the design of peer-to-peer networks was one of the hottest topics in systems research. Less so these days, but peer-to-peer networks remain practically relevant and also provide an intriguing case study of how a system evolves in response to incentive issues.

The story begins in 1999, with Napster. The primary purpose of Napster was to provide a centralized and searchable directory listing which users have copies of various files (e.g., mp3s). Prior to this, Internet users exchanged files in a more ad hoc fashion, for example via newsgroups. Napster's role was that of a match-maker: it matched up people who wanted a file with people who already have it. The file transfer itself was then done directly between the matched users. Note that a user generally acts as a server for some file transfers, and as a client for others (in contrast to the traditional client-server model).

In 2000, lawsuits against Napster started rolling in (for copyright infringement). Most famously by the RIAA, but also directly by artists, including Metallica (pissing off all their fans) and Dr. Dre. After Napster failed to comply with requests to block access to copyrighted material, they were shut down in 2001.

Napster's spectacular rise (up to around 25M users) clearly demonstrated the demand for peer-to-peer networks. Its shutdown by the government highlighted the vulnerabilities of a centralized system (like Napster's directory), so the next step was to develop completely decentralized peer-to-peer networks. In the new systems, each user maintains connections to a group of peers, and searching for a file now involves searching through the network of connections (by breadth-first search, in effect). Once a user with the desired file is located, the file transfer is done directly between the two users. There were several such systems; a major one was Gnutella, which debuted in 2000.[1] The fully decentralized nature of this

*©2016, Tim Roughgarden.

†Department of Computer Science, Stanford University, 474 Gates Building, 353 Serra Mall, Stanford, CA 94305. Email: `tim@cs.stanford.edu`.

[1]Others at that time included the now-forgotten FastTrack and eDonkey. Many clever follow-up designs followed throughout the mid-aughts.

second generation of peer-to-peer networks made them basically impossible to shut down.

Gnutella already highlights the kind of incentive issues that plague peer-to-peer networks. The functionality of Gnutella rests on (at least some) users conforming to the following intended behavior: upon receiving a file request, either upload the file to the requester (if the user has the file) or forward the request onward to peers (if not). But users were given no incentives to actually behave in this way.

Adar and Huberman [1], in the paper "Free Riding on Gnutella," showed that the incentive issues in Gnutella were real.

Vocabulary Lesson

free rider (n.): someone who enjoys a costly good or service without paying for it.

In the context of Gnutella, a free rider is a user who downloads but never uploads. The study in [1] showed that free riding was the dominant behavior in Gnutella, with over two-thirds of the users free riding, and with 37% of the uploads performed by only 1% of the users. These findings do not suggest a very functional or sustainable system—at some point, the uploaders will get fed up with all the free riders and find something better to do with their time and upload capacity. Indeed, in a 2005 follow-up study [4], the free-riding rate had climbed to 85%. In the subsequent years, Gnutella faded to black (so to speak).

2 The Prisoner's Dilemma

We next digress into some basic game theory, to explain a clean and famous way to think about incentives issues like free riding in Gnutella.[2] Game theory will teach us some lessons in a simple model, and then we'll return to peer-to-peer networks to apply these lessons.

Consider the following *file transfer game*. There are two "players," Alice and Bob. Alice possesses a file requested by Bob, and vice versa. Alice and Bob simultaneously decide whether or not to upload the requested file. (Each makes their own independent decision.) For each player, the benefit of receiving the desired file is 3, and the cost of uploading the requested file is 1 (due to bandwidth charges or opportunity costs).

There are four possible outcomes of the file transfer game, and these can be neatly summarized via the following "payoff matrix."

	Upload	Don't Upload
Upload	$2, 2$	$-1, 3$
Don't Upload	$3, -1$	$0, 0$

In each entry of the matrix, the first number is the payoff to the "row player" (here, Alice), and the second the payoff to the "column player" (Bob). Naturally, players want to maximize their payoffs. So is it better for the players to upload, or not?

[2]In this course, we'll draw on game theory largely through examples. Game theory is famous for examples that fit on a cocktail napkin and yet accurately capture the essence of incentive issues that are ubiquitous in the real world.

The answer depends on whether we consider what's good for the individual, versus what's good for the collective. From an individual's perspective, it's a no-brainer to not upload—you receive a higher payoff than if you do upload, no matter what the other player does. The technical term for this is a *dominant strategy*.

Vocabulary Lesson

dominant strategy (n.): a strategy that always maximizes a player's payoffs, no matter what other players do.

When the players choose their dominant strategies, neither one uploads and both receive a payoff of 0.[3] This outcome is not Pareto optimal—if both players upload, then both are strictly better off (with a payoff of 2).

The file transfer game is a rephrasing of a game known as the *Prisoner's Dilemma*. The usual story is: two people are corralled by the police and accused of a crime. The police interrogate the two suspects separately, and try to get each suspect to rat out the other one. If each prisoner accuses the other, then both receive a lengthy sentence. If neither prisoner accuses the other, they receive only a light sentence for a lesser charge. If only one prisoner accuses the other, then the turncoat is set free and the accused is locked up for life. The structure of these payoffs is the same as in the file transfer game. In the Prisoner's Dilemma, the strategies are usually called "cooperate" (corresponding to uploading) and "defect" (don't upload).

The Prisoner's Dilemma isolates the essential conflict that can arise between incentives for an individual and for a collective. It shows up all the time in real life, and it's useful to be able to recognize it. We've already seen that free riding in Gnutella can be interpreted in terms of the Prisoner's Dilemma. The unraveling of two-sided markets, with firms making ever-earlier offers (Lecture #2), stems from the same kind of problem. What other real-life examples can you think of?[4]

3 The Repeated Prisoner's Dilemma

Free riding in Gnutella shows that the incentive issues highlighted by the Prisoner's Dilemma are real, and that the individual incentive to defect can be powerful. On the other hand, it's easy to think of real-world examples of the Prisoner's Dilemma where the players seem to cooperate. For example, two competing companies deciding whether or not to spend massive amounts of money on advertising can be thought of as a Prisoner's Dilemma-type situation, and yet in many cases we do not see overly costly advertising wars (DraftKings vs. FanDuel notwithstanding). For another example, there are some two-sided markets that seem to resist unraveling (like most graduate school admissions processes). How can we explain this?

[3]This outcome is a special case of a *Nash equilibrium*, which is an outcome in which each player plays a best response to the other player. That is, neither player can increase their payoff by unilaterally switching strategies.

[4]Student suggestions in lecture included doping in sports, and peacocks competing to grow ever-more-cumbersome tails to attract the best mates.

3.1 Models of Repeated Play

One intuition is that, in many real-world settings, the short-term gain from defecting is outweighed by its long-term costs. Capturing this intuition with a mathematical model is subtle.

Repeated Prisoner's Dilemma (Model #1)

- Alice and Bob play the Prisoner's Dilemma games n times, for some known $n \geq 1$.
- The overall payoff to a player is the sum of his or her payoffs across the n games.
- The action chosen by a player in a stage i can depend on the past history (who played what in the first $i - 1$ stages).

One might hope that as n grows large, cooperation would emerge. But if we apply cold, clinical logic to this model, we'll see that this is not the case.

First, think about the last stage, stage n. At this stage, there is no future, and hence no future consequences of a defection. So just like in the one-shot Prisoner's Dilemma, it is a dominant strategy for each player to defect in stage n (no matter what happened previously). Rewinding to stage $n - 1$, we see that the action played at this stage will have no effect on the future (at stage n, both players will defect anyways). So again, given what's going to happen in the future, in stage $n - 1$ it's a dominant strategy for each player to defect. Continuing backward, we find that the only justifiable behavior in this model is for both players to always defect.[5]

The take-away of this exercise is not that cooperation in real-world settings is somehow "irrational." Rather, it points out that we need a mathematical model different from the most obvious one. In effect, we're taking as "ground truth" that cooperation can be sensible in repeated Prisoner's Dilemma-type settings, and trying to reverse engineer a mathematical model consistent with this belief.

The key idea for our second model is to have a *random number* of stages.

Repeated Prisoner's Dilemma (Model #2)

1. Alice and Bob play the Prisoner's Dilemma.
2. With probability p, the game stops. With probability $1 - p$, return to step 1 and play another stage.

Payoffs are again added across however many stage games are played, and the action chosen by a player at some stage can again depend on the past. We assume that a player wants to maximize his or her expected payoffs (where the expectation is over the randomness in when the repeated game stops).

[5]This type of argument is sometimes called *backward induction.*

3.2 The Emergence of Cooperation

We claim that if p is sufficiently small, then cooperation makes sense. (Note that if $p = 1$, then we're just back to the single-shot Prisoner's Dilemma.) For example, suppose Alice plays the "grim trigger" strategy.

Grim Trigger Strategy

- if Bob ever defected in the past, then defect;
- otherwise, cooperate.

Thus if Bob ever defects, Alice punishes him forevermore.

What is Bob's best strategy? If Bob ever defected in the past, then Alice's future behavior is fixed so Bob may as well defect forevermore. So the interesting case is where Bob has already cooperated in the first $i - 1$ stages ($i \geq 1$)—should he continue to cooperate in stage i? Here's the cost-benefit analysis:

	stage i	later stages
cooperate	2	$\geq (1-p)2$
defect	3	≤ 0

To explain, note that Alice will cooperate in stage i (since Bob never defected in the past). Thus at stage i, Bob will get payoff 2 for cooperating and 3 for defecting. If Bob defects at stage i, then Alice will defect at all future stages and the best expected payoff Bob can get from these stages is 0 (if he also always defects from now on). Suppose Bob cooperates at stage i. There will be a stage $i+1$ with probability $1-p$, and if there is one, Bob will receive a payoff of at least 2 at that stage (because Alice will cooperate in that stage). Thus the expected payoff from future stages is at least $(1 - p)2$. The expected payoff of cooperating exceeds that of defecting provided play continues with more than 50% probability (i.e., $p < \frac{1}{2}$).

The grim trigger strategy is pretty extreme, what with holding a grudge in perpetuity for a single infraction by the other player. The next strategy involves not only punishment, but forgiveness.

Tit-for-Tat Strategy

- in stage 1, cooperate;
- in stage i, do whatever the other player did in stage $i - 1$.

The Tit-for-Tat strategy starts optimistically, punishes immediately, and forgives quickly. This turns out to be a good strategy for playing the repeated Prisoner's Dilemma (and perhaps life?).

For example, if both players use the Tit-for-Tat strategy, then both players always cooperate. Note that the initialization is crucial: if both players use a strategy that differs only in defecting in the first stage by default, then the result is mutual defection at every stage.

Just as with the grim trigger strategy, if Alice plays Tit-for-Tat, and $p < \frac{1}{2}$, then the best response for Bob is to cooperate at every stage. The short-term gain of defection (1) is outweighed by the expected cost at the next stage $(2(1-p))$.

Our mathematical story for why cooperation appears in the repeated Prisoner's Dilemma is now reasonably satisfying: in real life, the number of repetitions is usually unknown (random, in effect) rather than known; and the Tit-for-Tat strategy resembles plausible real-life behavior in such situations.

3.3 The Axelrod Tournaments

The Tit-for-Tat strategy plays a starring role also in experiments on the repeated Prisoner's Dilemma. Around 1980, Robert Axelrod invited friends and colleagues to enter into a tournament for computer programs playing the repeated Prisoner's Dilemma [2]. There were 15 contestants, and each program played the other 14 programs in a repeated Prisoner's Dilemma game with 200 stages (so, a round-robin tournament). The payoff of a program was averaged over all 200 stages of all 14 matches. The winning entry (submitted by Anatol Rapoport) was—you guessed it—Tit-for-Tat. Tit-for-Tat was also the shortest entry, and many of the other programs were (trying to be) sophisticated. What makes this all the more remarkable is that Tit-for-Tat cannot win a head-to-head match with any opponent (Exercise Set #3)! But it scored highly against everybody.

Axelrod circulated the results of the first tournament and solicited entries for a second tournament with the same rules. This time, there were 62 entries. Rapoport re-submitted Tit-for-Tat, completely unchanged, and won again! Even though other programs were explicitly tailored to exploit Tit-for-Tat, they imploded when pitted against each other.[6]

3.4 Lessons Learned

Summarizing, the Prisoner's Dilemma starkly illustrates an essential conflict between individual incentives and the collective good. Many real-world settings are plagued by this conflict. Without long-term interactions between players, the incentive to defect is powerful and real, with free-riding on Gnutella being just one example. With repeated play, however, the long-term benefits of cooperation often outweigh the short-term benefits of defection. The Tit-for-Tat strategy appears particularly good for the repeated play of the Prisoner's Dilemma.

How can we apply these lessons to the design of peer-to-peer networks?

[6] Apparently no one submitted the program that can only improve over Tit-for-Tat (against every possible opponent): play Tit-for-Tat, except at stage 200 always defect.

4 BitTorrent

4.1 The Protocol

The *BitTorrent* protocol is currently the dominant paradigm in peer-to-peer file distribution. In 2013, BitTorrent was responsible for around 80% of the total peer-to-peer traffic (see [6, Chapter 5] for references). For example, you might have used it to download the latest version of Linux.

The BitTorrent protocol is designed to solve the problem of distributing a large file (a movie, an OS, a big software patch, etc.) to many users. Some BitTorrent terminology: a *swarm* is a group of users who have or want the same file (or pieces of it, see below). A *tracker* keeps track of the currently active users in a swarm. Trackers were originally implemented in a centralized way. Mindful of the dangers of centralization (recall Napster), tracking is now often done in a decentralized way (using a “distributed hash table (DHT),” for example). A user contacts the tracker when it wants to join a swarm, and can request from the tracker a list of a random subset of other users (perhaps 50 of them) in the swarm. The tracker serves only to coordinate; data transfers are done directly between users.

A file is broken into *pieces*, and users do transfers at the granularity of pieces (not of the entire file). Numbers vary, but a representative example would be a 10 GB file, broken into a thousand 10 MB pieces. There are a number of motivations for breaking a file into pieces and transferring only one piece at a time, such as the more efficient use of network capacity. But the most interesting justification is to shift the incentive structure in the file transfer game from that of the single-shot Prisoner’s Dilemma (at the level of files, as in Gnutella) to that of the *repeated* Prisoner’s Dilemma, with stages roughly corresponding to pieces. Downloads can often require tens of minutes or even hours, and involve hundreds or thousands of pieces. Thus, users have the opportunity to monitor the behavior of other users, and act accordingly.

4.2 Default Settings

Next we describe the reference client for BitTorrent; its implementation can be regarded as the intended behavior by users, according to the creators of the BitTorrent protocol. Key points of this implementation include:

1. A user re-announces itself to the swarm’s tracker every 15–30 minutes, requesting a new subset of users each time. This generally results in the number of peers of a user growing over time.

2. Peers regularly exchange information about who has which pieces of the file.

3. A user attempts to download simultaneously from all of its relevant peers, where “relevant” means that the peer has at least one piece of the file that the user doesn’t have.

4. For a parameter s, a user uploads to s of its peers, allocating a $1/s$ fraction of its upload capacity to each. A typical choice for s is 4, though larger values (10, 25, etc.) are also often used. A user is sometimes said to have s *slots*.

5. The lucky s peers to upload to are chosen using a variation of the Tit-for-Tat strategy.[7] Precisely, the user uploads to the s peers from whom it downloaded the most data in the past time period (e.g., 15 seconds). That is, if a peer was especially nice to the user in the previous stage, then the user will reciprocate.

6. When uploading to a peer that wants several of a user's pieces, priority is given to the rarest piece in the user's neighborhood (i.e., the piece possessed by the fewest number of the user's peers).[8]

Actually, this description isn't quite accurate. Maybe you already spotted the bug (hint: it has nothing to do with incentives). The issue is that there's no way for a new user to get started. What happens if you're new to a swarm? You presumably have no pieces whatsoever of the file. Thus, you are not a "relevant peer" for any other user, and no one will ever upload to you, and your plight will be neverending.

For this reason, the actual reference client includes "optimistic unchoking." (A peer that a user is uploading to has been *unchoked*; the rest are *choked*.) While the user reserves $s-1$ slots to be allocated using the above Tit-for-Tat strategy, it uses its sth slot to upload to a random neighbor (which is re-chosen regularly, such as every 45 seconds). This can function as a "pity upload" to a user who doesn't have much of the file yet. Once a user gets lucky and is optimistically unchoked a few times, it will have enough pieces to contribute usefully to the swarm.[9]

4.3 BitThief

Users are free to use a BitTorrent client other than the reference one. Can you think of any incentive issues in the reference client? Are there clever ways for a user to deviate from the default settings that lead to better results for the user?

The goal of the unapologetically named *BitThief* client [5] is to complete a download without ever uploading anything, just like the free riders on Gnutella. (Perhaps to avoid legal consequences, or to save money, etc.) If all other users are using the reference client, then the only way to do this is to get the entire file via optimistic unchokes. To grow its neighborhood and hence get optimistic unchokes as quickly as possible, the BitThief client pesters the tracker incessantly, asking it for more users much more frequently than the default of every 15–30 minutes. Downloads with BitThief were slower than with the reference client, but not unusably so (5x slower was typical).

[7] This is not just a metaphor; Bram Cohen, the creator of the BitTorrent protocol and its reference client, really was inspired by the repeated Prisoner's Dilemma and the Tit-for-Tat strategy [3].

[8] Rare pieces are potential bottlenecks to a user completing its download in a reasonable amount of time. This is a heuristic to hasten downloads.

[9] In typical downloads using BitTorrent, progress is quite slow at the beginning.

There is a pretty easy way to mitigate the "incentive attack" of BitThief: just implement the tracker so that it ignores repeated requests from the same IP address in a 30-minute sliding window.

4.4 BitTyrant

The point of the *BitTyrant* client [7] is to be smarter than the reference client when it comes to allocating its upload capacity. The motivation is again to improve performance for the user, but unlike in BitThief, here it is not obvious whether or not the deviation causes harm to the rest of the system. (E.g., if all users used BitThief, no data would ever be transferred!)

The key idea in BitTyrant is to have each user i maintain an estimate d_{ij} for the download capacity it would get from a peer j if j unchoked i, and also an estimate u_{ij} for the amount of upload capacity i would need to spend to get j to unchoke it. If j has unchoked i in the past, then d_{ij} and u_{ij} can be derived from past interactions. Otherwise, some ingenuity is required. One way to estimate d_{ij} is for i is to first estimate j's total download capacity using the rate at which it acquires new pieces (recall peers keep each other informed about their inventories of pieces), then assume that j's upload capacity is the same as its download capacity, and then make an assumption about the number s of slots that j is using (e.g., $s = 4$). One way to estimate u_{ij} is to keep uploading more and more data to j (e.g., increasing by a multiplicative factor of $1 + \alpha$ each time period, for small $\alpha > 0$) until j unchokes i.

Given its current estimates d_{ij} and u_{ij} for all peers j, user i allocates its upload capacity in a natural greedy way (like in the knapsack problem). Specifically, peers are sorted in nondecreasing order of "return on investment" d_{ij}/u_{ij}, and upload capacity is awarded to peers in this order (until the capacity is used up), with u_{ij} capacity being awarded to peer j. Note that the number of slots used by i is not fixed a priori, and different peers might get different amounts of upload capacity (unlike in the reference client). Also note that there are still incentive issues with BitTyrant—how would you game the system if you knew that everyone else was using BitTyrant?

Do you have any ideas for building a still better BitTorrent client?

References

[1] E. Adar and B. A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), 2000.

[2] R. Axelrod and W. D. Hamilton. The evolution of cooperation. *Science*, 211:1390–1396, 1981.

[3] B. Cohen. Incentives build robustness in BitTorrent. In *Proceedings of the First Workshop on Economics of Peer-to-Peer Systems*, 2003.

[4] D. Hughes, G. Coulson, and J. Walkerdine. Free riding on Gnutella revisited: the bell tolls? *IEEE Distributed Systems Online*, 6(6):1–18, 2005.

[5] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free riding in BitTorrent is cheap. In *Proceedings of the Fifth Workshop on Hot Topics in Networks*, pages 85–90, 2006.

[6] D. C. Parkes and S. Seuken. Economics and computation. Book in preparation, 2016.

[7] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in BitTorrent? In *4th USENIX Symposium on Networked Systems Design & Implementation (NSDI)*, pages 1–14, 2007.