Understanding and Improving Piece-related Algorithms in the BitTorrent Protocol

Jiaqing Luo, Bin Xiao, Kai Bu, and Shijie Zhou

August 19, 2012

Abstract

Piece-related algorithms, including piece revelation, selection and queuing, play a crucial role in the BT protocol, because the BT system can be viewed as a market where peers trade their pieces with one another. During the piece exchanging, a peer selects some pieces revealed by neighbors, and queues them up for downloading. In this paper, we provide a deep understanding of these algorithms, and also propose some improvements to them. Previous study has shown that the piece revelation strategy is vulnerable to under-reporting. We provide a game-theoretic analysis for this selfish gaming, and propose a distributed credit method to prevent it. Existing piece selection strategies, though long believed to be good enough, may fail to balance piece supply and demand. We propose a unified strategy to shorten the download time of peers by applying utility theory. The design of the piece queuing algorithm has a conflict with that of piece selection strategy, because it is not possible to assume that the queued requests for a selected piece can always be available on multiple neighbors. We give a possible fix to address the conflict by allowing peers to dynamically manage their unfulfilled requests. To evaluate the performance of the proposed algorithms, we run several experiments in a live swarm. Our primary results show that they can achieve fast individual and system-wide download time.

Chapter 1

Introduction

BitTorrent [2] (BT) is one of the most common Peer-to-Peer (P2P) file sharing protocols, which accounts for more than roughly 27%-55% of all Internet traffic as of February 2009 [15]. Piece-related algorithms are core parts of the BT protocol, because the BT system relies upon peers to cooperatively trade their *pieces* with one another. When exchanging pieces, peers need three algorithms for different tasks. First, they tell neighbors which pieces they have, called *piece revelation*. After that, they decide what pieces to download, called *piece selection*. Finally, they scatter requests on multiple neighbors, called *piece queuing*. Our understanding of these algorithms is still far from complete, because some fairly basic questions have, to date, gone unanswered: Why is the piece revelation strategy vulnerable to selfish gaming? Are existing piece selection strategies really good enough? Why is the design of piece queueing algorithm still under dispute? In this paper, we provide some answers to above questions in order for a clear guideline for developers, and a better system performance in terms of *individual good* – fast download time of an individual peer, and *social good* – fast average download time of all peers.

The first contribution of our work is to provide a game-theoretic analysis for piece revelation, and propose a *distributed* credit method to prevent selfish gaming. Experimental results [11] have shown that *under-reporting*, where peers do not disclose the pieces they truly have, may degrade the overall performance. However, there are no clear explanations for the performance degradation. Our study finds that under-reporting can enhance peer's competitiveness in non-cooperative games. In particular, a peer who under-reports its pieces can gain some extra interest from its neighbors, because it is able to upload some pieces that should be uploaded by its opponents. However, when all peers under-report their pieces to others, there may be an overall performance degradation, because each peer will spend more time on downloading from low-speed neighbors. To encourage neighbors to reveal as many pieces as they truly have, we suggest that peers work cooperatively to credit their neighbors by examining *have* messages.

The second contribution of our work is to apply utility theory to piece selection, and design a *unified* strategy to balance piece supply and demand. Nowadays, two strategies are well-known in piece selection. One is *random* in which peers choose

to download pieces in a random order. While a random piece can enrich the diversity of piece supply, it might not meet the needs of neighbors. The other is *rarest-first* in which peers download pieces in a rarest first order. Though the rarest piece can make a peer more attractive to others, it may only be available from very few neighbors, which makes it slow to download. Moreover, it might not fulfill emergent needs of some neighbors although it is in a high demand. To improve the efficiency of piece sharing, we propose a *utility-driven* strategy, which covers both rarest-first and random, by constructing two utility functions – *neighbor's utility* and *peer's utility*. Rather than selecting a random or rarest piece, we suggest that a peer selects the one that can maximize its utility.

The third contribution of our work is to address the conflict in piece queueing, and revise existing algorithm to *dynamically* mange unfulfilled requests. As pointed out by some BT protocol specifications [5, 16], there exists a conflict between piece selection and piece queuing. On the one hand, a peer tends to select a rarest piece to make itself more attractive to others [5] On the other hand, it expects to queue the requests for a selected piece at each neighbor to fully utilize others' bandwidth [16]. Unfortunately, the requests for a selected piece are not always spread over multiple neighbors. To reduce memory consumption and increase bandwidth usage, we design a *dynamic scatter* algorithm in which peers separate queued pieces into two groups – *rational* and *sub-rational*. By adjusting the ratio between the two group, peers can dynamically control the size of the list to store queued pieces.

To evaluate the performance of the proposed algorithms, we implement a java-based client, and run several experiments in a live swarm. The primary results show that the proposed algorithms can work better than original ones, and achieve both individual good and social good.

The rest of the paper is organized as follows. Chapter 2 gives a brief introduction to the BT system, and describes the goals of this work. Chapter 3 provides a new insight into under-reporting. Chapter 4 designs a distributed credit method to prevent under-reporting. In Chapter 5, we consider various factors in piece selection, and show rarest-first may fail to achieve its design goal. Chapter 6 proposes a utility-driven strategy to shorten the download time of peers. In Chapter 7, we design a dynamic scatter algorithm to resolve the dispute in piece queuing. Chapter 8 presents experimental results of the proposed algorithms. Finally, Chapter 9 concludes this paper.

Chapter 2

Background

In this chapter, we firstly give a brief introduction to the BT system. We then describe the goals in this work – individual good and social good, and make some reasonable assumptions – rational peers and fair incentives.

2.1 BT system basics

A BT system commonly consists of *trackers*, *seeders* and *leechers*. A *tracker* keeps track of peers who are participating in the process of downloading and/or uploading a particular file, and makes this information available to others who want to download the file. Peers in the BT system are either *seeders* who have a complete file and are willing to serve it to others, or *leechers* who are still downloading the file and are willing to serve the pieces that they already have to others. Before joining a *swarm* which is a peer group sharing a particular file, a peer firstly downloads a .torrent file from a web server, which contains a URL list of trackers and other information related to the sharing file. After that, the peer makes an HTTP GET request to a tracker on the URL list. Upon receiving the request, the tracker randomly returns a subset of peers sharing the file. Then, the peer attempts to initiate TCP connections with the returned peers, which will become its *neighbors*.

Files in the BT system consist of *pieces* (32-256KB in size) which in turn consist of blocks (16KB in size). A block is a portion of data that a peer requests from neighbors. A peer is *interested* in a neighbor, if the neighbor has at least one piece that it does not have. A peer's interest in one of its neighbors is based solely on what pieces that neighbor claims to have. Uploading in the BT system is called *unchoking* which dictates to whom and how much to unchoke. Every 10 seconds, a peer unchokes only 4 neighbors who have the highest upload speed and are interested in downloading from it, called *regular unchoking*. This maximizes the peer's download rate. To try out unused connections, at least 1 random neighbor is unchoked by the peer regardless of that neighbor's contribution, called *optimistic unchoking*. A peer determines the rarity of pieces by keeping an initial *bitfield*, which is a bit array of its pieces, from each neighbor, and updating it with every *have* message. Based on these bitfields, the peer

downloads pieces in a random or a rarest-first order from neighbors.

2.2 Goals and assumptions

We decompose our goal in this work into two subgoals. Our fist goal is to resolve the disputes in piece-related algorithms for making the BT protocol more clear. Our second goal is to improve the performance of piece-related algorithms for achieving both individual good and social good.

The BT system can be viewed as a market where peers exchange their pieces with one another. The term *interest* reflects piece supply vs. piece demand, because a peer sends only to those interested in it, and equivalently, receives only from those in whom it is interested. The efficiency of piece sharing will increase, if *all* peers can maximize the number of neighbors who are interested in them. We assume that all peers are *rational*, that is, they are smart enough to predict others' decisions, and behave to garner maximum neighbor interest in a unit cost. We further assume that BT's incentive is fair such that peers can get more if they upload more. The current used incentive, unchoking, is fair to some extent, because it originated from Tit-for-Tat (TFT), which imitates the mechanism of living creature evolving and the nature choosing [8]. We do not assume cooperative peers, that means, peers can choose either cooperative or noncooperative strategies. However, we would like to point out that individual maximization may not lead to socially optimal outcome [7]. To achieve social good, peers should cooperate instead of compete.

Chapter 3

Study of piece revelation

In this chapter, we describe possible strategies that can be used for piece revelation, and provide a game-theoretical analysis, and explain why the BT protocol is vulnerable to under-reporting.

3.1 Piece revelation strategies

Generally, a peer has three strategies to reveal pieces to neighbors: over-reporting, full-reporting and under-reporting (including non-reporting). According to the BT protocol, a peer should honestly disclose all its pieces to make others clearly know the rarity of pieces, called *fully-reporting*. However, a selfish peer has an incentive to strategically manipulate others into helping it download faster. To keep neighbors interested in it, a selfish peer can exaggerate the pieces it has, called *over-reporting*, or hide some pieces it has, called *under-reporting*. In economics and contract theory, these strategic manipulations can be characterized by *information asymmetry* where some parties have an information advantage over others [7].

Over-reporting can be easily detected, because a peer cannot respond to the requests for the pieces that it falsely claimed. Compared with over-reporting, under-reporting works more stealthily. A selfish peer only reveals a piece to a neighbor when the neighbor is going to lose interest. As providing a neighbor with a rare piece would make the neighbor potentially removing some interest from it, it reveals the most common piece it has that the neighbor does not. Experimental results [11] have shown that under-reporting will cause 12% overall performance loss when it becomes widespread.

Although previous work [11] has revealed that under-reporting is a potential threat to BT's fairness and performance, some questions are left unanswered: why does under-reporting degrade the overall performance? What kind of swarms are more vulnerable to under-reporting? In the following sections, we provide some answers to above questions through a game-theoretical analysis.

3.2 Under-reporting in heterogeneous swarms

We design a 2-player game to show that under-reporting causes more overall performance degradation in a heterogeneous swarm than in a homogeneous swarm. In such a game, players (peers) a and b upload their pieces to another peer ℓ. Each player has two strategies *Fully report* and *Under-report*. If a player plays *Fully report*, it reveals all its pieces to ℓ. One the other hand, if the player plays *Under-report*, it won't reveal the rarest piece until ℓ has completed the most common piece. Both players want to maximize their payoffs in terms of ℓ's interest measured in time slots. As shown in Figure 3.1, a has pieces 1 and 3, b has pieces 2 and 3. If a and b play *Under-report*, they will reveal the most common piece 3 first.

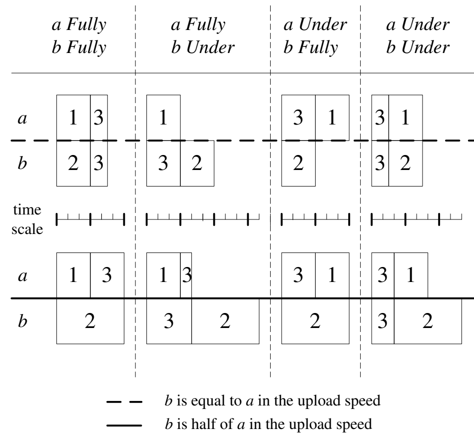

Figure 3.1: Upload processes, showing that a and b upload to ℓ.

Without loss of generality, we consider two cases. In Case A, b has the same upload speed as a. Assume that the upload speed for each player is one piece per time slot, we show players' payoffs of Case A in Figure 3.2. We then have three observations. 1) For both players, *Under-report* strictly dominates *Fully report*, called *dominant strategy*. Therefore, (*Under-report*, *Under-report*) is a strict *Dominant Strategy Equilibrium*, which is a set of strategies in which every player is playing a dominant strategy 2) *Under-report* could make a player gain some extra interest from ℓ, because the player is possible to upload some pieces that should be uploaded by its opponent. For example, if both a and b play *Fully report*, a can upload a half of piece 3. However, if b changes its strategy to *Under-report*, a has no chance to upload that piece. 3) The outcome of (*Under-report*, *Under-report*) is the same as that of (*Fully report*, *Fully report*). This implies that *Under-report* may have a limited effect on overall performance in a homogeneous swarm where players have similar upload speeds.

In Case B, b has a lower upload speed than a. Assume that b is half of a, we illustrate players' payoffs of Case B in Figure 3.3. We then have two observations. 1) *Under-report* is a weakly dominant strategy for a, while it is a strictly dominant strategy for b As we can see that if b plays *Fully report*, a is indifferent between

		Player *b*	
		Fully report	*Under report*
Player *a*	*Fully report*	*3/2, 3/2*	*1, 2*
	Under report	*2, 1*	*3/2, 3/2*

Figure 3.2: The payoffs of Case A where b is equal to a in the upload speed.

playing *Under-report* and playing *Fully report*. However, a never plays *Fully report* because it is rational and knows that b will never play *Fully report*. 2) The outcome of (*Under-report*, *Under-report*) is not good for the high-speed player, which could result in a degradation of overall performance, because each peer would spend more time on downloading from low-speed players. For example, ℓ's download time increases from 2 to $\frac{8}{3}$.

		Player *b*	
		Fully report	*Under report*
Player *a*	*Fully report*	*2, 2*	*4/3, 10/3*
	Under report	*2, 2*	*5/3, 8/3*

Figure 3.3: The payoffs of Case B where b is half the upload speed of a.

Within the game-theoretic analysis, we can draw some useful conclusions. 1) A rational player always plays *Under-report*. 2) *Under-report* is a non-cooperative strategy that could enhance peer's competitiveness. However, the non-cooperative equilibrium does not necessarily mean the best payoff for each player. 3) *Under-report* may degrade overall performance in a heterogeneous swarm where peers have different upload speeds.

3.3 Under-reporting in large swarms

We use an n-player game to illustrate that peers are more likely to adopt under-reporting (or non-reporting) when swarm size becomes larger. In this game, n players know each other in a swarm (a fully connected swarm of n peers). Each player has some pieces

that others do not have, and would like to share pieces with others but prefers that someone else reports its pieces. Suppose each player attaches the value v to a piece being reported (thereby the piece being shared), and bears a cost c of reporting the piece. The value v could represent the prolonged interest for the reporter (it uploads to others), or a newly complete piece for others (they download from the reporter). The cost c could state the bandwidth consumption of broadcasting *have* messages and the potential risk of losing others' interest. We assume $v > c > 0$. Player i has two strategies, *Report* and *Not Report*, where $\forall i \in \{1, 2, ..., n\}$. The payoff of player i, u_i, is defined as follows:

- $u_i(\text{Report}) = v - c$
- $u_i(\text{Not Report, at least one other player reports}) = v$
- $u_i(\text{Not Report, none of the other players report}) = 0$

Figure 3.4 illustrates players' payoffs of a 2-player game. We have two pure-strategy Nash equilibria, where one player plays *Report* and the other plays *Not report*. Both are asymmetric, where different players play different strategies. For the n-player game, we have n pure-strategy Nash equilibria, where a player i plays *Report*, and the remaining $n - 1$ players play *Not report*. But again these are asymmetric, and hence, do not resolve the issue of which of the players play *Report*. For the symmetric equilibrium, we look for the one involving mixed strategies. Let p_i be the probability that player i plays *Report*. Since we are trying to find the symmetric equilibrium, it must be true that at equilibrium, all players will be reporting with the same probability. Let that be p^*. Without lose of generality, we use the indifference condition for player i.

$$\begin{aligned}
&u_i(\text{Report}) = u_i(\text{Not Report}) \\
&\Rightarrow v - c = 0 \cdot Prob(\text{no one reports}) + v \cdot Prob(\text{someone reports}) \\
&\Rightarrow v - c = v\big(1 - Prob(\text{no one else reports})\big) \\
&\Rightarrow v - c = v\big(1 - (1 - p^*)^{n-1}\big) \\
&\Rightarrow p^* = 1 - \big(\frac{c}{v}\big)^{\frac{1}{n-1}}
\end{aligned}$$

Since $v > c > 0$, p^* decreases when n increases. This indicates that, if n is large, players tend to play *Not report*, because they are more likely to believe that someone else will play *Report*.

		Player *b*	
		Report	*Not report*
Player *a*	*Report*	*v-c, v-c*	*v-c, v*
	Not report	*v, v-c*	*0, 0*

Figure 3.4: Two-player illustration of Peer's Dilemma.

Chapter 4

A distributed credit method

In this chapter, we design a distributed credit method to detect and punish under-reporters, and propose some improvements to make the method more practical.

4.1 Design and limitation

The basic idea of the distributed credit method is that three fully connected peers, called a *basic component*, credit each other to detect under-reporters by examining *have* messages. When a normal peer has complete a new piece, it will broadcast a *have* to all its neighbors, which contains the index of that piece. On the other hand, an under-reporter may stay silent to hide the piece information. If a peer, called *assessor*, wants to credit one of its neighbor, called *assessee*, it will check whether the assessee's neighbor, called *intermediator*, receives a *have*, after it completes uploading a piece to the assessee. If the intermediator gets the *have*, the assessor will increase the assessee's credit. Assuming peers who are honest can earn credits more easily than peers who are dishonest, by earning credits peers can signal their honesty to others. A similar idea can be found in the signaling game, which is a non-cooperative game with incomplete information [7]. In the job market, new employee signal their capacity for learning to prospective employers by finishing college.

To punish under-reporters, an assessor will take assessees' credits into account when unchoking. Figure 4.1 gives an example of the credit method. Peers a, b and c are fully connected. By exchanging neighborhood information, a and b can know that c is an overlapping neighbor. To test whether c is honest or not, a will inform b of a recent piece that c requested from it. Upon receiving a *have* from c, b will relay the *have* to a immediately. If the *have* does not arrives within a short time, a will suspect c to be an under-reporter, and clear the record of c's upload speed. As a result, c will rank at near bottom for the upload speed among a's neighbors. a will be less likely to regularly unchoke by c. This example shows that credit method does not change the rules of unchoking.

The limitation of the credit method is that a basic component may not be formed in a large swarm. Assume that the swarm size is n, and the number of peers returned

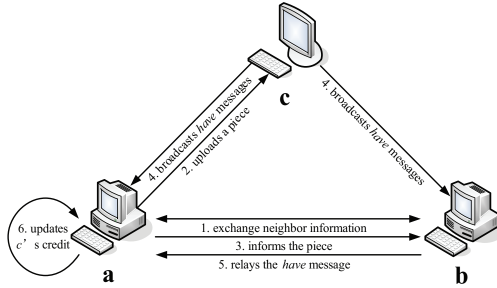

Figure 4.1: An example of the credit method, showing that a and b work together to credit c.

from a tracker is r. According to the BT protocol, connections between each peer pair are bidirectional and symmetrical. In other words, given any two peers i and j, if i is in the neighbor set of j, then j is in the neighbor set of i. There are two ways for connecting i and j. One is that j knows i from the tracker. This probability is $\frac{r}{n-1}$. The other is that j doesn't know i from the tracker, but i knows j from the tracker. The probability is $(1 - \frac{r}{n-1})\frac{r}{n-1}$. Considering these two cases, the probability that there is a connection between any two peers, $p_{\rightleftharpoons}$, is $\frac{r}{n-1} + (1 - \frac{r}{n-1})\frac{r}{n-1}$. Since peer connections are independent of each other, the probability that any three peers are fully connected, $p_{\triangle}$, is $\left(\frac{r}{n-1} + (1 - \frac{r}{n-1})\frac{r}{n-1}\right)^3$. In most cases, $p_{\triangle}$ is higher than 0.029, because r is 50 by default [16, 3], and n is less than 300 for more than 95% swarms [17].

The credit method is a simple yet robust method to throttle against under-reporting. However, it may not be very effective in a large swarm due to the perceived low $p_{\triangle}$.

4.2 Improvement and configuration

A possible improvement for the credit method is to allow peers construct a basic component by the shallow flooding. The assessor will broadcast a *query* message with a small TTL (time to live) and the assessee's IP. The *query* will then propagate through the swarm hop by hop till TTL has expired. When receiving a *query*, a peer will check whether the assessee exists in its neighbor list. If not, it will decrease TTL by 1 and forward the *query* to all its neighbors. Otherwise, it will take the role of intermediator, and return a *queryhit* message containing its own IP. The *queryhit* will route back to the assessor along the inverse of the *query* path. By communicating with the intermediator, the assessor can create a basic component to credit the assessee. As shown in Figure 4.2, a tries to credit c, and sends a *query* to d. Since d doesn't know c, it forwards the *query* to b. As b is a neighbor of c, it returns a *queryhit*. After receiving the *queryhit*, a contacts b and starts the credit process. For fast and low-overhead transmissions, a connects to b using UDP instead of TCP.

The overhead of the credit method consists of two parts. One is the relayed *haves*.

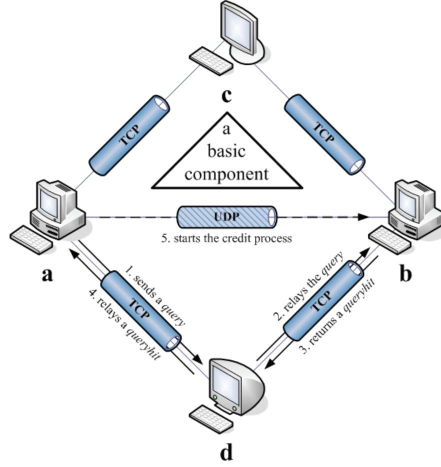

Figure 4.2: Credit process improvement, showing that a, b and c form a basic component.

The other is the *query* flooding. To reduce the overhead caused by relayed *haves*, we suggest that the credit method runs before each unchoking. In other words, it is called at a regular interval of 10 seconds. In each run, an assessor evaluates one or more assessees. For each assessee, it randomly chooses one intermediator to examine the *have*. Every run the evaluation of an individual assessee only requires one relayed *have*. To reduce the overhead caused by the *query flooding*, we determine the smallest value of TTL. By using the epidemic model [13], we compute the number of peers hit by *query* at each hop. The number of peers hit by *query* at hop 1, h_1, is $p_{\rightleftharpoons}(n-1)$. The probability that a peer is not hit at hop 2 is $(1-p_{\rightleftharpoons})^{h_1}$. Thus, the number of peers hit at hop 2, h_2, is $\big(1-(1-p_{\rightleftharpoons})^{h_1}\big)(n-1-h_1)$. By analogy, the number of peers hit at hop TTL, h_{TTL}, can be written as:

$$h_{TTL} = \big(1-(1-p_{\rightleftharpoons})^{h_{TTL-1}}\big)(n-1-\sum_{i=1}^{TTL-1} h_i) \tag{4.1}$$

Suppose that the *query* aims to hit at least n_{hit} peers, the smallest value of TTL, TTL_{min}, can be determined by the following inequalities:

$$n_{hit} > \sum_{i=1}^{TTL_{min}-1} h_i, \quad n_{hit} \leq \sum_{i=1}^{TTL_{min}} h_i \tag{4.2}$$

By using the controlled shallow flooding, a peer can credit any neighbor with low overhead.

4.3 Robustness evaluation

The credit method may face some security challenges, because it requires some information reported by the third peer. We evaluate the robustness of the credit method by analyzing the following two malicious behaviors.

Relaying denied

The assessee is the common neighbor of intermediator and assessor. If the assessor chokes the assessee, it will upload fewer pieces to the assessee, and rank lower for upload speed in assessee's neighbors. The intermediator will have more chances of being unchoked by the assessee due to the lower ranking the assessor. Thus, a selfish intermediator will intentionally not relay the *have* to fool the assessor. Fortunately, each time the assessor randomly chooses one intermediator to credit the assessee. Suppose that the assessee has N neighbors, the probability that a selfish neighbor is selected to be the intermediator is $\frac{1}{N}$. If the assessor executes the credit method c times, the selfish neighbor, one average, can fool the assessor $\sum_{k=0}^{c}\binom{c}{k}(\frac{1}{N})^k(1-\frac{1}{N})^{c-k}k$ times. For example, let $N = 80$ (the maximum number of neighbors for a peer), and $c = 120$, the selfish neighbor will only be selected 1.5 times. This result indicates that the gain of the selfish neighbor is limited, which may not affect BT's fairness.

Collusion

A selfish assessee has an incentive to collude with other peers, called *collaborators*, to relay forged *haves* to make the assessor believe that it has fully reported its pieces. The collaborators may not share pieces with the assessee, but try to pass themselves off as neighbors of the assessee. They will respond a *queryhit* to the assessor, when they receive a *query*. The assessor can judge whether an assessee is selfish or not according to the number of *queryhits*, because a selfish assessee typically has an unusual large neighbor set.

Chapter 5

Study of piece selection

In this chapter, we describe existing strategies that are used for piece selection. We then consider various factors affecting piece supply and demand to explain why rarest-first may fail to achieve its design goal of maximizing neighbor interest.

5.1 Piece selection strategies

In marketing, a firm needs to increase its stock to meet the market demand. Similarly, the primary goal of piece selection is to maximize neighbor interest in order to make the piece supply meet the demand.

The original BT protocol suggests that peers adopt a mixed strategy, called *standard*, which combines random, rarest-first and end-game [5]. When a peer has a few pieces, it selects pieces to download at random to quickly get as many complete pieces as possible so as to make others have an interest in it, called *random*. After it has more than 4 pieces, it prioritizes the download of pieces by their rarity to keep neighbors interested in it so that it can trade pieces using unchoking, called *rarest-first*. When the download is almost complete (the last piece is queued), it sends requests for all of its missing blocks to all of its neighbors to speed up the downloading, called *end-game*. To keep this from becoming horribly inefficient, the peer also sends a cancel to everyone else every time a block arrives. However, the updated BT protocol suggests that peers generally download pieces in a random order [3]. Our discussions will mainly focus on rarest-first for the following two reasons. One is that rarest-first is widely deployed in real-world systems, and is believed to be good enough [10]. The other is that peers use rarest-first in most time of their downloading.

Intuitively, the rarer the piece that a peer selects, the more interest it can gain. A question is raised here: Can rarest-first maximize neighbor interest? In the following sections, we show rarest-first is not good enough for piece selection by considering various factors influenced piece supply and demand.

5.2 Effect of download cost

Piece demand means not just how many peers download a given piece, but how many peers download that piece at its cost, and how many peers would download that piece if its cost changed. We give a simple example in Figure 5.1 to illustrate the effect of download cost on neighbor interest. For simplicity, we assume that neighbors are stable during the time period to select and download a piece, and, in each time slot, a peer only downloads one piece from one of its neighbors. In Example A, peer ℓ downloads slowly from neighbor a (a half piece per time slot), and the rarest pieces are 1 and 3, which are in shadow. ℓ prefers to select piece 3. Suppose all its neighbors select piece 1, ℓ will lose all neighbors' interest at the second time slot, because it cannot complete piece 3 before its neighbors complete piece 1. A rational choice for ℓ is piece 4 rather than piece 3, because a complete piece can make it more attractive to others. Choosing piece 4, ℓ can prolong a's interest.

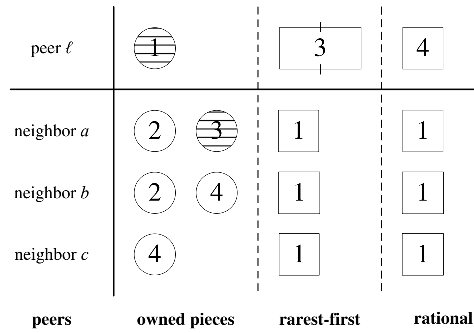

Figure 5.1: Example A: ℓ will lose all neighbors' interest if it selects the rarest piece 3.

5.3 Effect of neighbor surplus

Neighbor surplus, occurring when quantity supplied is more than quantity demanded, is an important factor for peers to consider when entertaining thoughts about changing services that they offer. We give another example in Figure 5.2 to show the effect of surplus pieces on neighbor interest. In Example B, ℓ downloads from each neighbor at an equal speed, and pieces 3 and 4 are rarest, which are in shadow. ℓ would like to choose piece 4. Suppose all its neighbors select piece 3, ℓ will lose a's interest at the second time slot. An interesting observation is that, if ℓ chooses piece 5 rather than 4, it will maintain mutual interest with all its neighbors. The reason is that either b or c has 2 surplus pieces from ℓ (ℓ can provide 2 pieces to either b or c), while a only has 1 surplus piece from ℓ.

For a further explanation, we quantity a neighbor's interest to ℓ by a utility function on the number of surplus pieces from ℓ. If ℓ selects piece 5, it will prolong a's interest. The number of surplus pieces for a will increase from 1 to 2. If ℓ chooses 4, it will

garner interest from both b and c. For each of them, the number of surplus pieces will go up from 2 to 3. The total utility increases along with the number of surplus pieces. Usually, we do not look at the total utility but at the *marginal utility* which denotes the increase in utility for one more surplus piece. Following *the law of diminishing marginal utility* [14], the marginal utility usually decreases as the number of surplus pieces increases. In other words, the utility of selecting piece 5 may be higher than that of selecting piece 4. We will give a more detailed discussion in Section 6.1, after we define the utility functions.

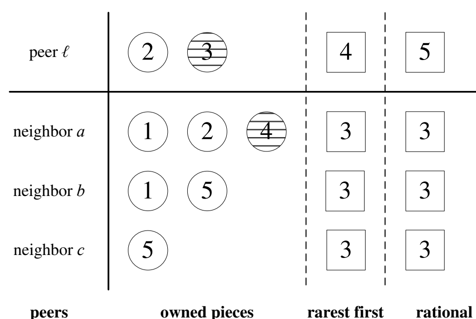

Figure 5.2: Example B: ℓ will lose a's interest if it selects the rarest piece 4.

5.4 Effect of piece rarity

One of the biggest advantages of rarest-first might be the "high" data availability that avoids *local* rare pieces from appearing. Here's an interesting question: what is the role of piece rarity in piece selection? We provide some answers to this question as follows. From a protocol perspective, maximizing interest, which depends on several factors (e.g., download cost, neighbor surplus and piece rarity), could make unchoking work more effectively as well. Piece rarity is just *one* factor, but not *all*. From a sharing perspective, peers only share complete pieces with others. Rarest-first cannot ensure fast piece completion time. From a measurement perspective, data availability might not be a critical problem in the BT system, though the system could be highly dynamic. Many real-word measurements [10, 18, 9] suggest that, in most swarms, there exists a significant number of seeders. Many peers stay for a few hours after their download is complete, and some peers even stay for days or weeks [4]. From an overhead perspective, it is very costly to let each peer to have global knowledge of others. To be practical, each peer makes its own decision based on a limited view of the whole system. There is no guarantee that the *local* rarest piece will be the *global* rarest one. Overall, piece rarity has an important role in piece selection, but it might not be as important as what we used to believe. Rather than *always* selecting the rarest pieces, we suggest that a peer selects them only if needed.

Chapter 6

A utility-driven strategy

In this chapter, we propose a utility-driven strategy to maximize neighbor interest based on the factors analyzed in the above chapter (e.g., download cost, neighbor surplus and piece rarity). We also discuss rules for choosing utility functions, and give some analysis of existing strategies.

6.1 Design of the utility-driven strategy

We assume that peer ℓ has N neighbors, and the file shared contains P pieces. We use n_i and p_j to represent the i-th neighbor and j-th piece respectively, where $\forall i \in \{1, 2, .., N\}, \forall j \in \{1, 2, ..., P\}$. If ℓ is not choked by n_i, ℓ will select and download a piece p_j if necessary. We use three terms, utility, cost and preference, to describe how a utility-driven strategy is made.

Neighbor's utility

As mentioned before, if n_i does not have p_j, ℓ can potentially prolong n_i's interest by downloading p_j. To determine whether n_i has p_j or not, we define an indicator variable $\alpha_i(p_j)$:

$$\alpha_i(p_j) = \begin{cases} 0 & \text{if } n_i \text{ does not have } p_j \\ 1 & \text{if } n_i \text{ has } p_j \end{cases} \quad (6.1)$$

From the viewpoint of n_i, there are two parameters, d_i and e_i, that may affect n_i's interest to ℓ. 1) d_i is n_i's full download speed. The larger d_i is, the sooner n_i loses its interest in ℓ. By observing the rate of *have* messages sent by n_i, ℓ can roughly estimate d_i. 2) e_i stands for the number of surplus pieces for n_i. The less e_i is, the more likely n_i is to become uninterested in ℓ. By considering these two parameters, we simply define n_i's utility function on e_i as a power function.

$$U_i(e_i) = -\frac{d_i}{e_i} \quad (6.2)$$

By Equation 6.2, we can measure n_i's interest to ℓ. We stress that this utility function can be re-defined according to different application requirements. We will discuss the rules for choosing them in Section 6.2. Suppose that n_i does not have p_j, namely, $\alpha_i(p_j) = 0$, e_i will increase by 1, if ℓ downloads p_j. The marginal utility $MU_i(e_i)$ can be calculated as:

$$MU_i(e_i) = \frac{d_i}{e_i(e_i + 1)} \tag{6.3}$$

From Equation 6.3, we see that an additional e_i provides smaller and smaller increases in utility (*diminishing marginal utility*).

Download cost

Once ℓ selects p_j, it will split the piece into $\frac{S_p}{S_b}$ blocks, where S_p and S_b represents the size of piece and that of block, respectively. Then, ℓ requests blocks from the neighbors who have p_j and unchoke it. Here, we define another indicator variable β_i.

$$\beta_i = \begin{cases} 0 & \text{if } n_i \text{ chokes } \ell \\ 1 & \text{if } n_i \text{ does not choke } \ell \end{cases} \tag{6.4}$$

As blocks can be queued and downloaded in parallel, it is difficult to estimate the time cost to download p_j, denoted as $C(p_j)$. Roughly speaking, there are three parameters, $b_i(\cdot)$, $b_i(p_j)$, and u_i, that may have some effects on $C(p_j)$. 1) $b_i(\cdot)$ is the number of blocks queued at n_i. Before getting a new block from a neighbor, ℓ needs to wait responses for all blocks queued at that neighbor. 2) $b_i(p_j)$ denotes the number of blocks from p_j assigned to n_i. 3) u_i represents n_i's upload speed to ℓ. We capture the minimum $C(p_j)$ as follows:

$$\begin{aligned} C(p_j) &= min\Big(max\big\{\alpha_i(p_j)\beta_i S_b \frac{b_i(\cdot) + b_i(p_j)}{u_i}\big\}\Big) \\ s.t. &\sum_{i=1}^{N} b_i(p_j) = \frac{S_p}{S_b} \quad and \quad \forall i : b_i(\cdot) \leq \frac{S_p}{S_b} P \end{aligned} \tag{6.5}$$

The objective function tries to minimize the largest element of a set. Noice that elements in the set are not constant. This is a parallel scheduling problem which is known as a NP problem [6]. In this work, we simplify the calculation of $C(p_j)$ in order for easy implementation and real-time piece selection. In particular, we ignore the queued pieces, and assume uniform block assignment, namely, $b_i(\cdot) = 0$, and $b_i(p_j) = \frac{S_p}{S_b \sum_{x=1}^{N} \alpha_x(p_j)\beta_x}$. Then, we let $C(p_j)$ to be the average download cost:

$$\begin{aligned} C(p_j) &= \frac{\sum_{i=1}^{N} \alpha_i(p_j)\beta_i S_b \frac{b_i(p_j)}{u_i}}{\sum_{i=1}^{N} \alpha_i(p_j)\beta_i} \\ &= \frac{\sum_{i=1}^{N} \frac{\alpha_i(p_j)\beta_i S_p}{u_i \sum_{x=1}^{N} \alpha_x(p_j)\beta_x}}{\sum_{i=1}^{N} \alpha_i(p_j)\beta_i} \end{aligned} \tag{6.6}$$

Peer's preference

A peer's utility is associated with utility functions of all its neighbors, as interest from each individual neighbor is the basic unit for aggregating to its utility. This is very close to the concept of *social welfare* in welfare economics. We define ℓ's utility $U_\ell(p_j)$ as:

$$U_\ell(p_j) = \frac{\sum_{i=1}^{N} \big(1 - \alpha_i(p_j)\big) MU_i(e_i)}{C(p_j)} \tag{6.7}$$

In Equation 6.7, we consider both download cost and neighbor surplus. By considering the download cost, we can speed up piece completion time. Notice that the most common piece may not be the one with the lowest cost, if peers upload at different speeds. By considering the neighbor surplus, we can achieve a *win-win* cooperation between a peer and its neighbors. For the peer, it is less likely to be choked, because it can continuously contribute its upload. For the neighbors, most of them can find what they need from the peer. Thus, all peers can benefit from selecting the piece that maximizes $U_\ell(p_j)$.

Given any two pieces p_j and p_k for $j \neq k$, if $U_\ell(p_j) > U_\ell(p_k)$, ℓ prefers p_j to p_k, denoted as $p_j \succ p_k$. If $U_\ell(p_j) = U_\ell(p_k)$, ℓ is indifferent between p_j and p_k, denoted as $p_j \sim p_k$. In such a case, ℓ can select either of them.

Examples

We use examples discussed in Section 5 to show that a peer could make a rational choice if it employs the utility-driven strategy.

In Example A, we know that $u_a = \frac{1}{2}$, and $u_b = u_c = 1$. Assume ℓ only downloads from one of its neighbors, we have $C(p_3) = 2$ and $C(p_4) = 1$. If ℓ selects p_3, it will gain interest from b and c. We get $U_\ell(p_3) = \frac{\frac{2}{1 \cdot 2}}{2}$. If ℓ chooses p_4, it will attract a's attention. We have $U_\ell(p_4) = \frac{1}{1 \cdot 2}$. Thus, $U_\ell(p_3) = U_\ell(p_4)$, and $p_3 \sim p_4$. That means ℓ can choose either of them. But, ℓ will definitely choose p_3, if it employs rarest-first.

In Example B, we can see that ℓ has two pieces, p_2 and p_3. Since it already has p_2, a needs p_3 only. Other two neighbors need both p_2 and p_3. Thus, we have $e_a = 1$, $e_b = e_c = 2$. Suppose $C(p_4) = C(p_5) = 1$, if ℓ selects p_4, it will become more attractive to b and c. Then, we have $U_\ell(p_4) = 2\frac{1}{2 \cdot 3}$. If ℓ selects p_5, it will keep a interested. We get $U_\ell(p_5) = \frac{1}{1 \cdot 2}$. Therefore, $U_\ell(p_5) > U_\ell(p_4)$, and $p_5 \succ p_4$. ℓ will select p_5, which is the result expected.

6.2 Rules for choosing utility functions

We provide a guide to choosing utility functions based on function properties. When a peer cares more about data availability, it can set $U_i(e_i)$ to be a function with slowly diminishing $MU_i(e_i)$ (e.g., $U_i(e_i) = e_i$). In this way, $\alpha_i(p_j)$ will play an important role in $U_\ell(p_j)$, which reflects the rarity of pieces. The selected pieces will tend to be those that appear least frequently in bitfields. When the peer cares more about neighbor interest, it can set $U_i(e_i)$ to be a function with rapidly diminishing $MU_i(e_i)$ (e.g.,

$U_i(e_i) = -e_i^{-1}$ or $U_i(e_i) = -e_i^{-2}$). In doing so, e_i will become a main parameter in $U_\ell(p_j)$, which is the quantity of surplus pieces. The selected pieces will be biased to those that are likely to be requested in the future. Giving consideration to both data availability and neighbor interest, a peer can choose a function with moderately diminishing $MU_i(e_i)$ (e.g., $U_i(e_i) = e_i^{0.5}$ or $U_i(e_i) = \log(e_i)$). As mentioned earlier, we should emphasize more on neighbor interest, because maximizing interest is the prime goal of piece selection.

6.3 Analysis of existing strategies

We analyze existing strategies from two aspects: utility function and download cost.

We first look at the utility functions chosen by rarest-first and random, respectively. Since neither random nor rarest-first considers download cost, we assume that the download cost for each piece is constant. Rarest-first always selects the least common piece. Therefore, we have $U_\ell(p_j) = \sum_{i=1}^{N}(1-\alpha_i(p_j))$, namely, $MU_i(e_i) = 1$. We can simply set $U_i(e_i) = e_i$. Random treats all pieces equally. Hence, we get $U_\ell(p_j) = 0$, that means, $MU_i(e_i) = 0$. We can let $U_i(e_i) = 0$. Observing these utility functions, we find that both rarest-first and random pay little attention to neighbor surplus, because their $MU_i(e_i)$ is constant rather than diminishing. We also notice that random rarely considers piece rarity, because its $U_\ell(p_j)$ is also constant.

We then discuss the download cost to a selected piece in the two strategies. Assume that ℓ has no piece; all neighbors unchoke ℓ, and upload at an equal speed. The download cost of the rarest piece is $max\{\frac{S_p}{\sum_{i=1}^{N}\alpha_i(p_j)}\}$, while that of a random one is $\sum_{j=1}^{P}\frac{S_p}{P\sum_{i=1}^{N}\alpha_i(p_j)}$. Compared with the rarest one, a random piece may have a lower download cost.

Through the above analysis, we show that utility-driven covers both rarest-first and random, which can make the protocol more clear and flexible.

Chapter 7

Study of piece queuing

In this chapter, we describe the dispute in piece queuing, and give a possible fix to address the conflict.

7.1 A conflict in piece queuing

The design of piece queuing algorithm is still under dispute [16]. A peer tends to select a least common piece for more interest from neighbors, while it expects to queue the requests for a selected piece at each neighbor for the better use of bandwidth. If the number of queued pieces is configurable, it is not possible to assume that requests for a selected piece are always spread over multiple neighbors. The scatter algorithm suggests that a peer maintains a free list of blocks to be requested. Such an algorithm is simply a matter of going through the following steps:

1. When a neighbor connection needs to queue a new request, a peer will first search through the free list for the requests that can be satisfied by the neighbor.

2. If none can be found, it will select a piece using random or rarest-first, and then, divide the selected piece into requests and add them to the free list and finally select one of the newly added requests from the free list.

A flaw in the algorithm is that a peer may still fail to find a request that can be satisfied by the certain neighbor after step 2. Even though it can finally find one (assume the size of the free list is very large), the peer cannot predict how many pieces will be added into the free list. Figure 7.1 shows how scatter works. We see that b and c are uploading piece 3 to ℓ, while a is waiting for the requests from ℓ. If ℓ uses rarest-first, it will queue the rarest pieces 2 and 4, which are not available on a, before piece 1. We believe that it is not reasonable to insert too many pieces at a time, because, as time goes by, the queued pieces may become suboptimal to download. Besides that, the more pieces are queued for downloading, the more memory is allocated to incomplete pieces.

7.2 A dynamic scatter algorithm

We revise the scatter algorithm, called *dynamic scatter* to address the conflict in piece queuing. We suggest that a peer classifies queued pieces into two groups: *rational* and *sub-rational*. Rational pieces are selected from those are available on at least one neighbors, while sub-rational pieces are selected from those are available on the certain neighbor. By adjusting the ratio between the two, a peer can dynamically control the number of queued pieces. In particular, a peer directly selects a piece that the neighbor has in step 2. Notice that this selected piece is sub-rational. Then, it scans free list after step 2, and adds several rational pieces (e.g., the rarest pieces) if needed. Figure 7.1 illustrates how dynamic scatter works. We can see that ℓ will queue piece 1 which is available on a. Assume that the ratio between rational and sub-rational pieces is set to 1. If piece 3 is rational, the queuing process is done. Otherwise, ℓ will add two rational pieces 2 and 4.

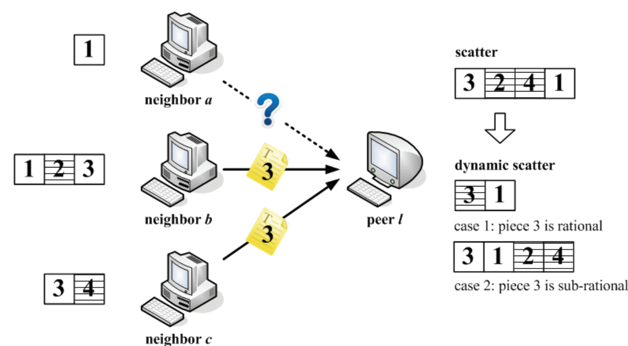

Figure 7.1: Queuing processes, showing that ℓ could reduce memory consumption by queuing piece 1.

The maximum number of queued pieces is bounded by that of unchoked neighbors For example, if the ratio is set to 1, the maximum number of queued pieces will not excess two times that of unchoked neighbors

Dynamic scatter ensures fast piece completion time because of the efficient use of neighbor bandwidth. In addition, it reduces the memory allocated to incomplete pieces, and satisfies the real time requirement in piece selection, because the number of queued pieces is not only configurable but also controllable.

Chapter 8

Performance evaluation

In this chapter, we evaluate the effectiveness and performance of the proposed algorithms through experiments. Our experimental results show that the proposed algorithms can achieve both *individual good* and *social good* when compared with previous well-known ones (e.g., rarest-first, random and scatter).

8.1 Experimental setup

We implemented a java-based BT client referring to [16, 5], and modified it to realize utility-driven and dynamic scatter. For research and testing purposes, we have uploaded the source code of our BT client to a web page at [12]. To evaluate the performance of our BT client, we set up an experiment to compare it with BitComet [1], a widely used BT client. Such an experiment ran over a private network consisting of 30 hosts. On 1 host, we installed a local tracker (using MyBT server), and started an initial seeder (using BitComet). On other 29 hosts, we installed either our BT client or BitComet. All peers would stay in swarm after they finished their downloading. There is no limit for peer's upload speed. The shared file is 152MB, which is divided into pieces of 512KB. From Figure 8.1, we can see that our BT client, which adopts both utility-driven and dynamic scatter, has similar performance to BitComet, which has made some optimizations that are not specified in the BT protocol (e.g., sequential I/O pattern). We can also see that dynamic scatter saves 83% average download time, and utility-driven further reduces 4% average download time. These results indicate that, in some cases (especially when there are a small number of initial seeders), it is preferable to select a downloadable piece than a mere potential of a rarer one.

We run the rest of our experiments in a controlled environment consisting of 5 hosts. Each host has 1.86GHz Intel Core 2 CPU, 2 GB RAM, and 1 GigE connection. Each host runs 1 BitComet as the initial seeder and 30 java-based clients as leechers. Unless otherwise specified, we use the following default parameters. Each peer has up to 80 neighbors, and uploads at 128KB/sec. Peers share a 20MB file which is fragmented into 128KB pieces. The maximum size of the free list is 10 pieces, namely, 80 blocks to be requested. Each peer leaves the swarm as soon as it has completed

its download. We let $U_i(e_i) = -e_i^{-1}$ as the default utility function for utility-driven. Each point in the figures that follow represents the average over at least 5 runs, and error bars denote 95% confidence intervals. Experiments on a large-scale platform or the PlanetLab testbed will be our future work.

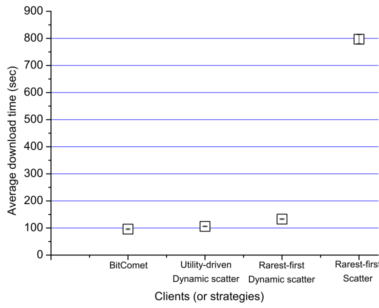

Figure 8.1: Different clients.

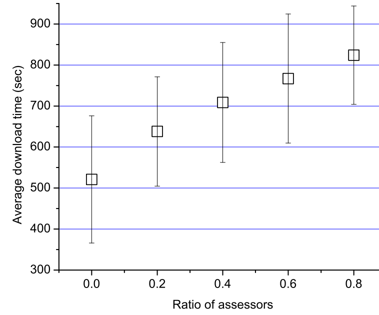

Figure 8.2: Punishment.

8.2 Effectiveness of the distributed credit method

Assume that all peers adopt standard and scatter. We start to show that the credit method increases the average download time of under-reporters. We randomly choose a peer to under-report its pieces, and vary the ratio assessors from 0 to 0.8. Each assessor credits all its neighbors by examining *have* messages. When a peer is suspected as the under-reporter, it will not be regularly unchoked by others. Figure 8.2 shows that the average download time of the under-reporter increases with the assessor ratio increases. The average download time increases by 58.2% when the assessor ratio increases from 0 to 0.8. We notice that two confidence intervals overlap. It is necessary to conduct a paired t test to determine whether the difference is significant. We set up two competing hypotheses referred to as the *null hypothesis*, which states that there is no significant difference, and the *alternative hypothesis*, which states that there exists a significant difference. Let X be the set of samples for the assessor ratio is 0, Y be the paired set of samples for the assessor ratio is 0.8, and n be the number of samples for each set. Given $X = \{570, 694, 466, 518, 357\}$ and $Y = \{660, 875, 827, 909, 849\}$, we calculate the mean and standard deviation of the differences, $\bar{d} = \frac{\sum_{i=1}^{n}(y_i - x_i)}{n} = 303$ and $s_d = \sqrt{\frac{\sum_{i=1}^{n}(d_i - \bar{d})^2}{n-1}} = 163.6$. Then, we calculate the standard error of the mean difference, $SE(\bar{d}) = \frac{s_d}{\sqrt{n}} = 73.2$. So, we have t-statistic, $t = \frac{\bar{d}}{SE(\bar{d})} = 4.1$. Under the null hypothesis, this statistic follows a t-distribution with $n - 1$ degrees of freedom. From the t-test table, we find that the critical value for a 2-tailed t with 4 degrees of freedom using the 0.05 significance level, $t_{0.05(4)} = 2.78$. As $|t| > t_{0.05(4)}$, we are inclined to reject the null hypothesis, and accept the alternative hypothesis.

Therefore, there is evidence that, on average, the credit method does lead to a noticeable performance degradation of the under-reporter. The reason for this is that the under-reporter downloads from others mainly through optimistic unchoking.

We then measure the overhead of the credit method caused by the *query* flooding. We expect that *query* covers 80% peers. This, we set n_{hit} to be 120, and increase TTL progressively. By Inequalities 4.2, we find that TTL_{min} is 2. The numbers of peers hit at hop 1 and 2 are 80 and 69, respectively. Figure 8.3 illustrates that overhead traffic increases along with the assessor ratio. When the ratio increases from 0.4 to 0.6, the overhead increases 44.8%. To reduce the overhead, assessors should broadcast *query* with a small TTL.

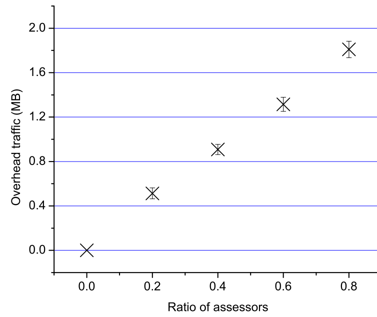

Figure 8.3: Overhead traffic.

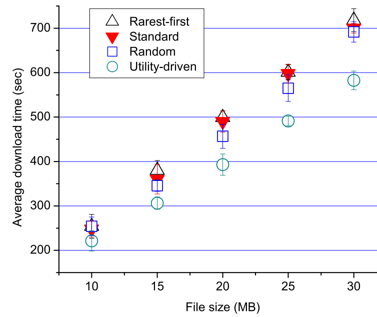

Figure 8.4: Different file sizes.

8.3 Performance of the utility-driven strategy

Assume that all peers adopt scatter. We compare utility-driven with other strategies, including rarest-first, standard and random. We also study the effects of various factors on system performance, such as file size, swarm size, upload speed, utility-driven ratio, maximum free list size, initial seeder ratio and seeding time.

Effect of the file size

Files shared in different swarms have different sizes. We set the file size to be $\{10, 15, 20, 25, 30\}$ MB. Figure 8.4 shows that utility-driven can save more average download time when the file size becomes larger. Compared with standard, utility-driven saves 10.9% and 16.9% system-wide average download time, when the file size is 10MB or 30MB, respectively. Another observation is that random outperforms both rarest-first and standard. Recall that only complete pieces can be selected and downloaded by others. Random outperforms standard because of its fast piece completion time. We notice that two confidence intervals overlap. Let X be the set of samples for random, Y be the set of samples for standard. When the file size is 15MB, we have $X = \{352, 360, 333, 338, 354\}$, and $Y = \{373, 374, 373, 351, 372\}$. Then, we get

$t = 4.39$, and have $|t| > t_{0.05(4)}$. Thus, we reject the null hypothesis. These results also indicate that standard and rarest-first have a similar average download time. Let X be the set of samples for standard, Y be the set of samples for rarest-first. When the file is 20MB, we have $X = \{499, 481, 494, 477, 491\}$, and $Y = \{514, 500, 503, 482, 482\}$. Then, we get $t = 1.61$, and find $|t| < t_{0.05(4)}$. As a result, we fail to reject the null hypothesis.

Effect of the swarm size

Most swarms consist of less than 300 peers. We set the swarm size to be {100, 150, 200, 250, 300}. Figure 8.5 depicts that utility-driven works efficiently in a large swarm. When the swarm size reaches 300, the average download time for utility-driven is 40.9%, 40.7% and 30.1% lower than that for rarest-first, standard and random, respectively. Utility-driven enhances the cooperation of peers, because it considers both neighbor interest and download cost.

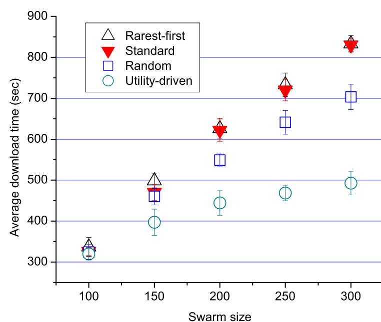

Figure 8.5: Different swarm sizes.

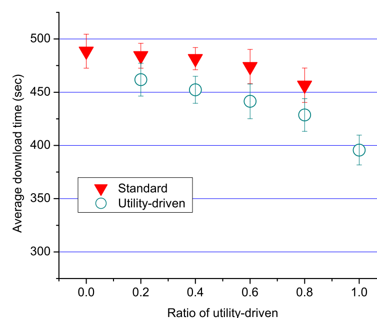

Figure 8.6: Different strategy ratios.

Effect of the utility-driven ratio

The deployment of utility-driven should not have any negative impact on existing systems. We vary the ratio between utility-driven and standard. Figure 8.6 shows that the trend of average download time is a downtrend when utility-driven becomes more popular. When the ratio increases from 0.6 to 0.8, the average download time for utility-driven and that for standard decrease by 3.8% and 2.9%, respectively. Another notable observation is that utility-driven is always better than standard. When the ratio is 0.6, the average download time for utility-driven is 7% lower than that for standard. As confidence intervals overlap, we calculate t and compare it with $t_{0.05(4)}$. Let X be the set of samples for utility-driven, Y be the set of samples for standard. We have $X = \{448, 441, 427, 447, 440\}$, and $Y = \{482, 459, 471, 471, 488\}$. Then, we get $t = 5.89$ and $|t| > t_{0.05(4)}$, and thus, reject the null hypothesis. The results show that peers who use utility-driven can perform well in a standard dominated swarm. Moreover, utility-driven is incrementally deployable. The more widely it is deployed, the better overall performance can be achieved.

Effect of the upload speed

Peers may upload at different speeds. We vary the maximum upload speed from 80KB/sec to 272KB/sec. From Figure 8.7, we can see that the benefit of utility-driven is more significant when peers upload at a higher speed. When the upload speed is up to 80, the average download time for utility-driven is close to that for random When the upload speed increases to 128KB/sec, the average download time for utility-driven is 10.4% lower than that for random These results indicate that utility-driven can better utilize the upload bandwidth of peers.

Effect of the initial seeder ratio

Mangy measurement results show that there are a lot of seeders in a swarm. We set the ratio of initial seeders to be {0.02, 0.04, 0.06, 0.08, 0.1}. One interesting observation from Figure 8.8 is that utility-driven is obviously faster than rarest-first when the ratio of initial seeders is 0.02. The reason for this is that rarest-first tends to select pieces that are only available on initial seeders, while utility-driven may select some *common* pieces that are available on multiple neighbors. Utility-driven can spread pieces faster over the swarm, when there is a small amount of initial seeders. Another interesting observation is that all strategies have a similar performance when the ratio of initial seeders increases to 0.1. The cooperation among peers becomes less important, because most peers can download from seeders directly.

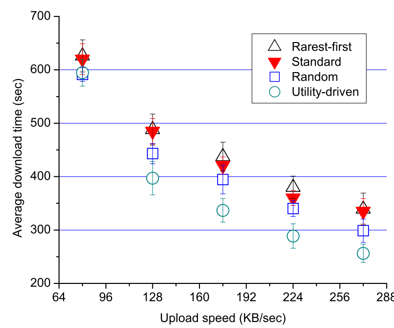

Figure 8.7: Different upload speeds.

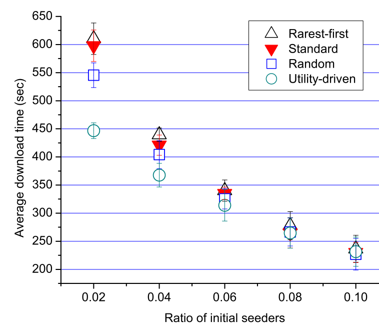

Figure 8.8: Different seeder ratios.

Effect of the seeding time

Many peers linger for a period of time after their download is complete, called *seeding time*. We vary the seeding time from 0 sec to 120 sec. Figure 8.9 shows that rarest-first is more affected by the seeding time When the seeding time increases from 0 sec to 120 sec, the performance of rarest-first and that of utility-driven increases by 23.5% and 16.8%, respectively. Although the experiment runs in a homogeneous swarm, peers will not finish their downloading at the same time due to the limited

service range of initial seeders (4 regular unchokings and 1 optimistic unchoking). A longer seeding time can reduce the download time of slow peers, and the download cost of rarest pieces.

Effect of the free list size

The maximum size of the free list is an important parameter but is not specified in the BT protocol. We set the maximum number of queued pieces to be {4, 8, 12, 16, 20}. From Figure 8.10, we see that standard works better when the maximum number of queued pieces increases. Compared with rarest-first, standard saves 2.6% and 3.8% average download time when the maximum number is 8 and 12, respectively. The reason is that a larger free list can make end-game work more efficiently. However, a larger free list consumes more memory, which may not be suitable for low memory computers.

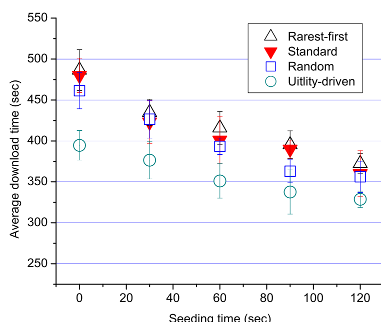

Figure 8.9: Different seeding times.

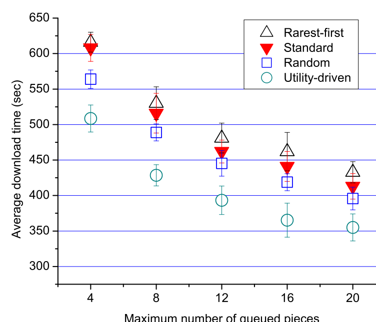

Figure 8.10: Different list sizes.

8.4 Performance of the dynamic scatter algorithm

Assume that all peers adopt standard. We begin to show that dynamic scatter can save the average download time of peers. As we discussed before, dynamic scatter will add several rational pieces to the free list, if the ratio between rational and sub-rational pieces is lower than or equal to a threshold. We vary the threshold ratio from 0 to 3, and choose utility-driven as the default strategy. Here, the threshold ratio 0 means that dynamic scatter will not add any rational piece, no matter how many sub-rational pieces have been selected. Figure 8.11 illustrates that dynamic scatter performs better than scatter. By setting the threshold ratio to 1, it reduces 35.7% system-wide average download time. To benefit the whole system, it is necessary for each peer to queue several rational pieces.

We then compare the memory cost of different algorithms. We record the queued piece in the free list. Figure 8.12 shows that dynamic scatter significantly reduces the average number of queued pieces When the threshold ratio is 1, the average number of

queued pieces for dynamic scatter is 38.1% lower than that for scatter. This indicates that dynamic scatter could be very useful when the BT protocol works over systems with low memory computers, like mobile phones. We suggest that the threshold ratio can be 1, because the average number of queued pieces increases along with the threshold ratio.

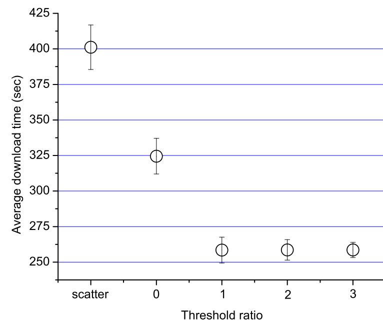

Figure 8.11: Different threshold ratios.

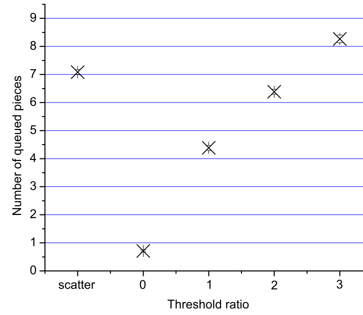

Figure 8.12: Memory consumption.

Chapter 9

Conclusion

In this paper, we dispel some myths about three piece-related algorithms, and propose some remedies for them as well. We propose a credit method to evaluate peer honesty by examining *have* messages, which are already been included in the BT protocol. The method is fully distributed, which requires no centralized server to operate. We propose a utility-driven strategy to enhance peer cooperation relying on two utility functions. Such a strategy is unified, win-win, feasible and deployable. In particular, it covers both rarest-first and random, all peers can benefit from it by serving others more and being served more, and requires neither a modification to the wire protocol nor a full deployment to benefit. We design a dynamic scatter algorithm to manage the size of the free list by dividing queued pieces into two groups, and adjusting the ratio between the two. This algorithm is dynamic, real-time and greedy. Specifically, it dynamically controls the number of queued pieces, meets the real-time requirement of piece selection, and fully utilizes each idle neighbor connection.

We implement the proposed algorithms in a java-based BT client, and run our experiments on a real swarm. Our primary experiment results show that they can outperform existing ones, and achieve both individual good and social good.

Bibliography

[1] BitComet, "A free c++ bittorrent/http/ftp download client." [Online]. Available: http://www.bitcomet.com/doc/changelog.php

[2] B. Cohen, "The bittorrent protocol specification." [Online]. Available: http://www.bittorrent.org/beps/bep_0003.html

[3] ——, "The bittorrent protocol specification," BitTorrent.org, Tech. Rep., 2008.

[4] R. R. Daniel Stutzbach, "Understanding churn in peer-to-peer networks," in *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2006, pp. 189–202.

[5] L. Fjeldsted, J. Fonseca, and B. Reza, "Specification and implementation of the bittorrent protocol," Tech. Rep., 2005.

[6] D. B. Fogel, "Applying evolutionary programming to selected traveling salesman problems," *Cybernetics and Systems*, vol. 24, no. 1, pp. 27–36, 1993.

[7] Fudenberg and Tirole, *Game Theory*. MIT Press, 1992.

[8] P. Gray, *Psychology*, 6th ed. Worth Publishers, 2010.

[9] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of bittorrent-like systems," in *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, Berkeley, CA, USA, 2005.

[10] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, New York, NY, USA, 2006, pp. 203–216.

[11] D. Levin, K. Lacurts, N. Spring, and B. Bhattacharjee, "Bittorrent is an auction: analyzing and improving bittorrent's incentives," in *SIGCOMM '08: Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, New York, NY, USA, 2008, pp. 243–254.

[12] J. Luo, "Supplementary files for the java bt client." [Online]. Available: http://www4.comp.polyu.edu.hk/~csbxiao/bittorrentweb/index.html

[13] J. Luo, B. Xiao, G. Liu, Q. Xiao, and S. Zhou, "Modeling and analysis of self-stopping btworms using dynamic hit list in p2p networks," in *SSN'09: IPDPS 2009 IEEE International Symposium on Parallel and Distributed Processing*, 2009, pp. 1–8.

[14] R. C. Merton, *Continuous-time Finance*. Blackwell, 1990.

[15] H. Schulze and K. Mochalski, "Internet study 2008/2009," 2009. [Online]. Available: http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf

[16] TheoryOrg, "Bittorrent protocol specification v 1.0," TheoryOrg, Tech. Rep.

[17] H. Wang, J. Liu, and K. Xu, "On the locality of bittorrent-based video file swarming," in *IPTPS'09: Proceedings of the 8th international conference on Peer-to-peer systems*. USENIX Association, 2009, pp. 12–12.

[18] X. Yang and G. de Veciana, "Service capacity in peer-to-peer networks," in *INFOCOM'04: Proceedings of the 23th Conference of the IEEE Computer and Communications Societies*, March 2004, pp. 2242– 2252.