University of Mississippi

eGrove

Electronic Theses and Dissertations | Graduate School

2014

Raptorqp2P: Maximize The Performance Of P2P File Distribution With Raptorq Coding

Zeyang Su
University of Mississippi

Follow this and additional works at: https://egrove.olemiss.edu/etd

Part of the Computer Sciences Commons

Recommended Citation

Su, Zeyang, "Raptorqp2P: Maximize The Performance Of P2P File Distribution With Raptorq Coding" (2014). *Electronic Theses and Dissertations*. 450.
https://egrove.olemiss.edu/etd/450

This Thesis is brought to you for free and open access by the Graduate School at eGrove. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of eGrove. For more information, please contact egrove@olemiss.edu.

RAPTORQP2P: MAXIMIZE THE PERFORMANCE OF P2P FILE DISTRIBUTION WITH RAPTORQ CODING

A Dissertation
presented in partial fulfillment of requirements
for the degree of Master
in the Computer Science and Information
The University of Mississippi

by

Zeyang Su

Apr 2014

Copyright Zeyang Su 2014
ALL RIGHTS RESERVED

ABSTRACT

BitTorrent is the most popular Peer-to-Peer (P2P) file sharing system widely used for distributing large files over the Internet. It has attracted extensive attentions from both network operators and researchers for investigating its deployment and performance. For example, recent studies have shown that under steady state, its rarest first scheme with the tit-for-tat mechanism can work very effectively and make BitTorrent near optimal for the generic file downloading process. However, in practice, the highly dynamic network environment, especially the notorious user churns prevalently existing in most peer-to-peer systems, can severely degrade the downloading performance.

In this thesis, we first study on the limitations of BitTorrent under dynamic network environments, focusing on two scenarios where with our preliminary modeling and analysis, we clearly identify how network dynamics and peer churns can significantly degrade the performance. With these findings, we further propose a novel protocol named RaptorQP2P, which is based on RaptorQ coding, to overcome the limitations of current BitTorrent design and maximize the performance of P2P file distribution. The new protocol features two levels of RaptorQ encoding. At the top layer, the entire file is RaptorQ encoded to yield a collection of source blocks and repair blocks, and then each source and repair block is RaptorQ encoded independently to yield a collection of source symbols and repair symbols for the block. The symbols are independently transferred among the peers and when a sufficient number of distinct symbols for a particular block have been received, whether source or repair, the block can be reconstructed. The file can be reconstructed using a sufficient arbitrary number of distinct blocks. Our results show that RaptorQP2P can well handle the network dynamics as well as peer churns and significantly shorten the downloading completion time by up to 41.4% with excellent scalability on both file size and user population.

DEDICATION

To my parents, Baohe Su and Huiming Dong

ZS

ACKNOWLEDGEMENTS

I would like to thank my advisors, Dr. Feng Wang and Dr. Yixin Chen , for guidance and patience throughout my graduate study at The University of Mississippi. I would also like to thank Dr. John N. Daigle, Dr. Byunghyun Jang, Dr. Haiyang Wang, Dr. Michael Luby, Dr. Sheng Liu, Dr. Zhendong Zhao for their assistance and advice.

I would like to thank my family for their encouragement and supporting me to pursue a degree in Master of Science.

TABLE OF CONTENTS

ABSTRACT ii

DEDICATION iii

ACKNOWLEDGEMENTS iv

LIST OF FIGURES vi

INTRODUCTION 1

RELATED WORK 5

PRELIMINARIES ON RAPTORQ AND BITTORRENT 7

ANALYSIS ON BITTORRENT AND ITS LIMITATIONS 12

RAPTORQP2P PROTOCOL DESIGN 17

PERFORMANCE EVALUATION 21

CONCLUSION AND FUTURE WORK 31

BIBLIOGRAPHY 33

VITA 36

LIST OF FIGURES

3.1 An overview of the BitTorrent Protocol. 9

4.1 An illustration for the Local View Scenario. 13

4.2 An illustration for two level encoding. 15

5.1 The algorithm for intelligent symbol scheduling. 19

6.1 Total downloading completion time as a function of file size for 1000 peers. . 24

6.2 CDF of individual peer's downloading completion time for a 512 MB file and 1000 peers. 25

6.3 Total downloading completion time as a function of the number of peers for a 512 MB file. 26

6.4 The contribution of opportunistic transmissions to the whole system as a function of file size with 1000 peers. 27

6.5 CDF of individual peer's getting first piece in a flash crowds condition for a 512 MB file and 1000 peers. 28

6.6 CDF of individual peer's getting first data for a 512 MB file and 1000 peers. . 29

6.7 Average time of getting first data as a function of the number of peers for a 512 MB file. 30

CHAPTER 1

INTRODUCTION

Peer-to-peer network which was surveyed by Liu et al. (2008) and Milojicic et al. (2002) is one of the most popular networks nowadays, attracting a large number of researchers to study on this area and propose to use it not only for supporting file sharing such as BitTorrent (2001) over the Internet, but also for video-on-demand which was discussed by Wu et al. (2011) and live streaming which was discussed by Wang et al. (2012a). There are two major advantages in a peer-to-peer network: flexibility and efficiency. Flexibility means peers can join and leave the system freely and the application cannot constrain such freedoms, while efficiency means each peer may contribute some capacity to the system, which can greatly relieve the load of a centralized server especially when the user number becomes large.

BitTorrent is the standard de facto for peer-to-peer file distribution, where its tit-for-tat and piece selection mechanisms have been heavily studied. The tit-for-tat is designed to prevent free-riders and help improve the fairness in the entire system. The piece selection mechanism which was discussed by Cohen (2003) includes four strategies : strict priority, rarest first, random first piece and endgame mode. The strict priority asks that before a piece is fully downloaded, all the data of that piece should be transferred from the same peer, unless that peer leaves the system or gets choked (i.e., being stopped due to a very low upload rate). And if the peer does not finish downloading a full piece, even it has some data of that piece, it still can not send these data to others. Rarest first requires that each peer first downloads the piece which is the rarest among its neighbors. Legout et al. (2006) showed that these mechanisms can help BitTorrent achieve near optimal performance when the network is under

steady state. However, Magnetto et al. (2010) and Spoto et al. (2010) report that in practice, the highly dynamic network environment, especially the notorious user churns prevalently existing in most peer-to-peer systems, can greatly degrade the downloading performance

In this thesis, we first take an in-depth study on the limitations of BitTorrent under dynamic network environments, investigating both local and global view scenarios where with our preliminary modeling and analysis, we clearly identify how network dynamics and users churns can significantly degrade the performance. For example, if one peer is receiving a piece from one of its neighbor, then even later a new neighbor with this piece and higher upload capacity comes, the new neighbor can do nothing to help speed up the downloading of this piece. Another example is that the optimality of rarest first can also hardly be achieved and maintained, as the peer joining and leaving by peer churns can easily change the piece availability distribution.

Motivated by these findings, we propose to use RaptorQ coding to overcome the identified limitations. RaptorQ coding which was proposed by Luby et al. (2011) is a kind of fountain codes which are also called rateless erasure codes or FEC codes. RaptorQ coding can generate as many symbols as desired on-the-fly (rateless codes), where each symbol is of equal value in decoding. Both encoding and decoding is in linear time. The new protocol features two levels of RaptorQ encoding, which is the most advanced technology in the series of practical implementations of fountain coding techniques. As will be explained in Section 3.1, in the standard RaptorQ encoding process, a file is partitioned into a collection of source blocks, and each of these source blocks are independently RaptorQ encoded to yield a set of source symbols and a potentially infinite number of repair symbols. The symbols, source and repair, are transferred across the network, and when a number of distinct symbols, whether source or repair, slightly exceeding the number of source symbols for a block has been received, the receiver is able to reconstruct that original block. For example, receiving two symbols more than the number of source symbols provides a decoding success probability of 99.9999%. In addition, the source of the encoded symbols is irrelevant. When all of the

blocks have been received, the entire file can be reconstructed.

In addition, we also exploit RaptorQ coding to increase the diversity and availability of pieces in the whole system, which can further improve the efficiency of downloading a regular piece as well as the last missing piece, which in practice, can often be a performance bottleneck of BitTorrent. The approach taken here is to apply an additional layer of encoding at the file level. That is, the original file is first used as input the the RaptorQ encoding process to yield a collection of source blocks and a potentially infinite number of repair blocks. The blocks, source and repair, are transferred using the standard RaptorQ process. Then when a number of distinct blocks, whether source or repair, slightly exceeding the number of source blocks of the file has been received, the receiver is able to reconstruct the entire file.

To the future use of RaptorQ as a vehicle for forming the units to be transferred among the peers, the new protocol also deals with the organization and rules for distributing the symbols. Many of the choices made in developing the present protocol resulted from an in-depth analysis of the performance of the leading peer-to-peer file sharing system, BitTorrent. It will be seen that the *source blocks* and *source symbols* of the RaptorQ approach are roughly equivalent to the *pieces* and *slices*, respectively, of the BitTorrent protocol.

We integrate these novel designs and develop a new protocol named RaptorQP2P. With the real world traces measured on the BitTorrent system, we conduct extensive simulations to evaluate our solutions. The results show that RaptorQP2P can well handle the network dynamics as well as peer churns and significantly shorten the downloading completion time by up to 41.4% with excellent scalability on both file size and user population.

The remainder of this thesis proceeds as follows: We review the related work in Chapter 2. Chapter 3 gives preliminaries of RaptorQ and BitTorrent. In Chapter 4, we model and analyze BitTorrent and its limitations under both the local view and global view scenarios, it is clearly identified how network dynamics and user churn can significantly degrade performance. And discuss how RaptorQ coding can overcome these limitations. We

propose our RaptorQP2P protocol in Chapter 5 and evaluate it by extensive trace-driven simulations in Chapter 6. we conclude the thesis in Chapter 7 with a discussion on the future work.

CHAPTER 2

RELATED WORK

Since the first version of BitTorrent was released on 2001, many researchers have been attracted to study in the peer-to-peer file distribution area. Cohen (2003) investigated the tit-for-tat and piece selection mechanisms, showing that the former can help prevent the free riding and the latter can make the diversity of the pieces in the whole system well balanced. Bharambe et al. (2006) used both analysis and experiments to show that BitTorrent may not always achieve the optimal, and the tit-for-tat in some situation may degrade the performance. Piatek et al. (2007) pointed out that the tit-for-tat can cause weak robustness.

Fountain Codes which was described by MacKay (2005) are also called rateless erasure codes. This coding technique divides the whole file into blocks and each block is then generated to rateless encoded symbols. When the receiver collects a certain number of encoded symbols, whose number is often greater than that of the original symbols, it can decode the symbols to the original block. Luby-Transform (LT) codes which proposed by Luby (2002) are the first generation of fountain codes, where the symbol length can be arbitrary and the encoded symbols can be generated on-the-fly. Recently, Raptor codes which was proposed by Shokrollahi and Luby (2011) were proposed as a type of more advanced fountain codes, which can conduct the encoding and decoding process in linear time. RaptorQ codes which was proposed by Luby et al. (2011) are the latest version of Raptor codes and can introduce even less decoding overhead (i.e., the number of extra symbols required to decode the original data).

Due to the good property of the fountain codes family, Luby (2012), Bouras et al. (2013) and Miguel et al. (2013) have been developed to use the fountain codes family to

improve the performance, where Eittenberger et al. (2012) proposed to utilize Raptor Codes to accelerate P2P streaming. There have also been studies on utilizing fountain codes to accelerate P2P file sharing. Spoto et al. (2010) proposed to modify BitTorrent by using the LT codes. The modification, however, may introduce duplicate coded packets being sent to the same peer, and even worse, a coded packet forwarded by a peer may traverse in a circle and then be forwarded back to this peer again. To address these issues and also make the data exchange more efficiently, Magnetto et al. (2010) proposed ToroVerde, a push-based P2P content distribution protocol, where a bloom filter is included in a coded packet when the packet is forwarded, so as to record the peers that the packet has traversed and avoid sending the packet to the same peer twice. However, the additional bloom filter introduces extra overhead to the data exchange. Also, when the peer number becomes large, the size of the bloom filter has to be increased so as to still distinguish different peers accurately, causing even more overhead for the data exchange.

Different from these works, we propose a new protocol design based on RaptorQ codes, which is one of the most recent advances in the Fountain codes family. Moreover, our design is based on the modeling and analysis of the limitations of BitTorrent under various dynamic network environments, which also guide us on the RaptorQP2P design to achieve the maximized performance for P2P file distribution.

CHAPTER 3

PRELIMINARIES ON RAPTORQ AND BITTORRENT

3.1 RaptorQ Coding

RaptorQ belongs to the family of Raptor codes which was proposed by Shokrollahi and Luby (2011) and is one of the recent advances on fountain codes which was introduced by MacKay (2005). Fountain codes are also called rateless erasure codes or FEC codes. LT codes which was discussed by Luby (2002) is the first generation of fountain codes, where the whole file is divided into many blocks and each block is further divided into source symbols. The source symbols will then XOR each other to generate encoded symbols. Ideally, the number of encoded symbols that can be generated is limitless. When the receiver receives a certain number of encoded symbols which is slightly greater than the source symbols, then the receiver can decode the symbols and get the block. Different from the LT codes, in RaptorQ, after we divide whole file into blocks and symbols, there is a pre-coding stage which the source symbols will XOR to generate some redundant symbols. These redundant symbols plus the source symbols will XOR each other and then generate encoded symbols, each encoded symbol having a specific symbol number within a block. After the receiver receives a prescribed number of encoded symbols , it can decode the block. The number of received encoded symbols should slightly more than the number of source number. For example if the number of source symbols are k, the receiver should receive $k + \varepsilon$ encoded symbols where ε is a very small number. By taking advantage of the pre-coding stage, the RaptorQ can achieve both encoding and decoding in linear time. And its decoding overhead can be much smaller than other Raptor codes and LT codes. In addition, a potentially infinite number of symbols can be generated on the fly and the number of symbols needed

by the receiver in order to decode is only slightly higher than the original number of source symbols, the source of the symbols being irrelevent. These properties are found to be very useful in designing a P2P file distribution protocol to deal with some of the challenging characteristics of P2P networks as will be shown later.

3.2 BitTorrent

Bram Cohen proposed the BitTorrent (2001) and designed the BitTorrent protocol in April 2001 and released the first BitTorrent version in July 2001. Different from the former HTTP or FTP client-server protocol, BitTorrent allows user uploads of file to others, which can greatly relieve the load of the file server and accelerate distribution of a large file to a large number of users. BitTorrent has become a standard protocol of file sharing over the Internet. To illustrate the protocol clearly, we first briefly introduce the terminology and then give an overview on the BitTorrent protocol.

3.2.1 Terminology

A user usually downloads a *.torrent* file from the web server, which includes metainfo of the file and the IP address of the tracker that can provide the information about other users currently downloading the file.

The *tracker* is a central server that collects statistics of all the users downloading a file. Each user should first contact with the tracker and then get a list of users to start downloading the file.

A *swarm* is the set of all the users that participate in distributing (i.e., downloading and uploading) a file.

Peers are the users who participate in distributing a file. The peers in a swarm may be divided into two types. One is the *leecher*, which does not have the whole file yet. The other is *seeder*, which has finished downloading but stays in the swarm to further help the file distribution. Only leachers will download the file content from other peers but both leachers

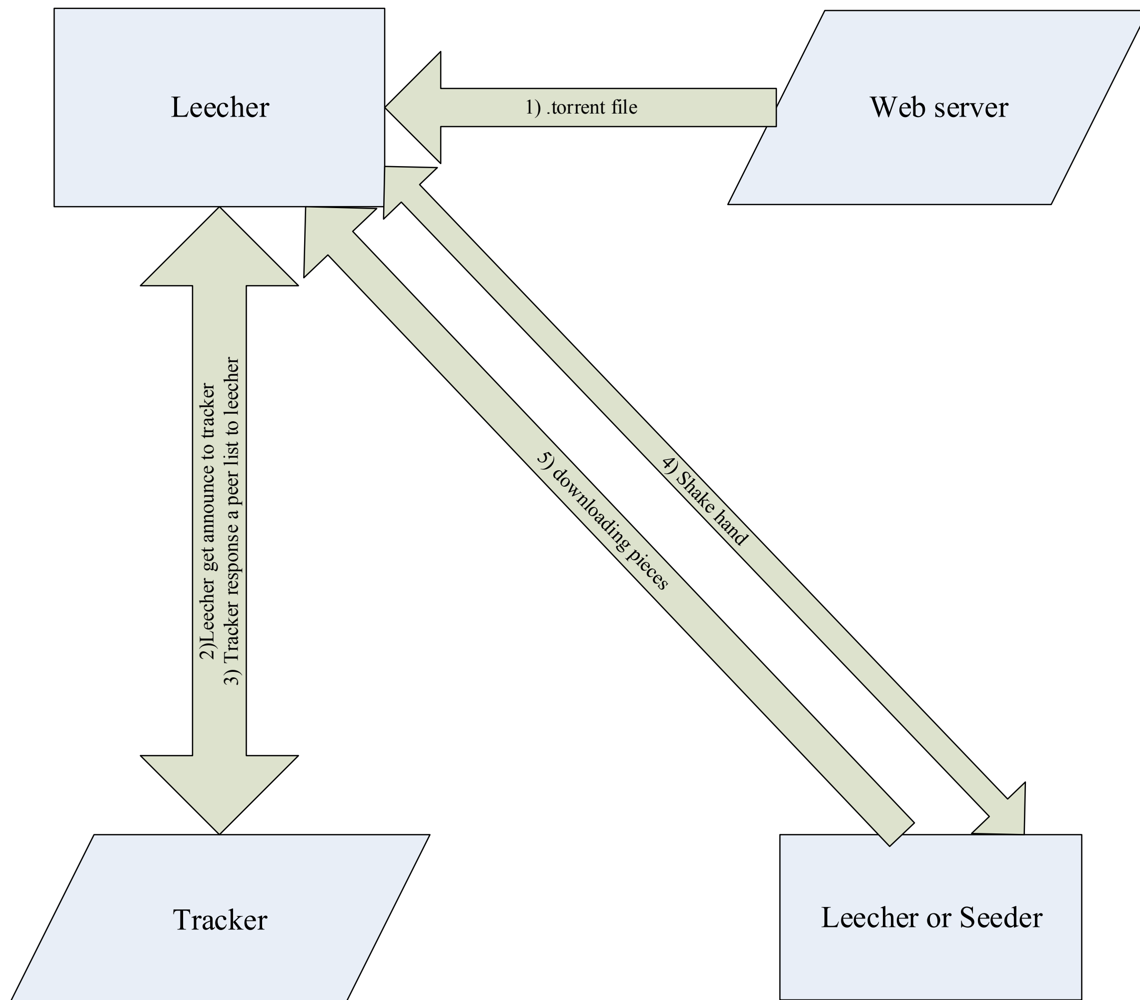

Figure 3.1. An overview of the BitTorrent Protocol.

and seeders can upload the file content to other peers. Each peer will maintain several peers as its neighbors.

In addition, BitTorrent divides the whole file into *pieces*. Each piece is further divided into *slices*. A slice is the smallest transmission unit in BitTorrent.

During the file distribution process, there is a *piecemap* in each peer to indicate its downloading progress.

If peer A does not have a certain piece but peer B has, then we say peer A is *interested* in peer B.

3.2.2 Protocol Overview

The mechanism of BitTorrent is shown in Fig. 3.1. One leecher first downloads a .torrent file from a web server. Then the leecher will contact with the corresponding tracker to ask for a list of peers currently in the swarm and the tracker will response the leecher a peer list. After that, the leecher will randomly select a number of peers from the list and connect to them for downloading pieces. When peers connect to each other, they will exchange their piecemaps to see whether a connected peer is interested or not, so that a peer can request a missing piece from others.

In BitTorrent, two mechanisms are used for piece requesting and downloading process. The first is the piece selection mechanism, which includes four parts: strict priority, rarest first, random first piece and endgame mode. Strict priority is that before a piece is fully downloaded, all the slices in that piece should be downloaded from only one peer, unless that peer leaves the system. If so, a second peer will be selected to download the remained slices of that piece. Also, a peer cannot share a piece with others until the whole piece has been fully downloaded. Rarest first means that when choosing a piece to download, a peer will select the piece which is the rarest among its neighbors. Random first is when a peer first joins the system, the peer will randomly download a piece from its neighbors. The endgame mode occurs when a peer only has one piece left to finish the downloading, in which case it will send the request to all its neighbors. Once a peer receives a response from one neighbor, it will download the last piece from that neighbor. Besides the piece selection mechanism, the second mechanism is called tit-for-tat, which is mainly designed for avoiding the free riders that only download from other peers but never upload to others. In tit-for-tat, a leecher will choke the peer with the lowest uploading rate, and a seeder will choke the peer with the lowest downloading rate.

These two mechanisms have been proved to be close to optimal if the network environment is at steady state. However, in practice, this assumption can hardly hold given the peer churns and the network dynamics, which may significantly degrade the overall system

performance. For example, by strict priority, for a given piece, a peer can only download it from one of its neighbors. This means that as long as that neighbor can upload the slices of the piece to the peer (i.e., not leave or get choked), the peer will stick to that neighbor for the piece, even if later a new neighbor with higher upload capacity enters the system and has the piece. Also, due to peer churns, the current rarest piece may change as time progresses since both peer joining and leaving can change the piece availability distribution, which is further aggravated by that the rarest first used in BitTorrent is from the local view instead of the global view. In next section, we will further analyze BitTorrent and its limitations under various network dynamics.

CHAPTER 4

ANALYSIS ON BITTORRENT AND ITS LIMITATIONS

In this chapter, we analyze and discuss how BitTorrent deals with various network dynamics and the limitations from both a local view scenario and a global view scenario, respectively. We then explain why RaptorQ codes can help overcome these dilemmas, which further motivates our RaptorQP2P protocol design proposed in next section.

4.1 Analysis on Local View Scenario

The local view scenario refers to the situation when a peer wants to download a piece from its neighbors. Fig. 4.1 shows an illustration, where peer A is currently neighboring with peers B, C, D and E. Suppose that peer A wants to download piece 1 and peers B, C and D have it. By strict priority, peer A can only choose one peer among B, C and D for downloading piece 1. However, no matter which neighbor peer A chooses to request the piece, it may become sub-optimal later due to the network dynamics and peer churns. For example, if peer A chooses B for currently B has the highest upload rate, then later due to the bandwidth variations, D may have higher upload rate than B yet A cannot switch to D for piece 1. In addition, if peer churns happen, say, peer C leaves and peer F becomes a new neighbor of A, then even F has higher upload rate than B, A still cannot switch to it for piece 1.

With the previous analysis, one may quickly work out an improvement where a peer is allowed to download a piece from several peers simultaneously, e.g., in Fig. 4.1, Peer A downloads piece 1 from B, C and D simultaneously. To achieve the optimal performance, we thus have the following modeling and analysis. Let $up(x)$ denote the upload rate of peer x. Assume a piece can be further evenly divided into m slices $\{b_1, b_2, \ldots, b_m\}$ with size s.

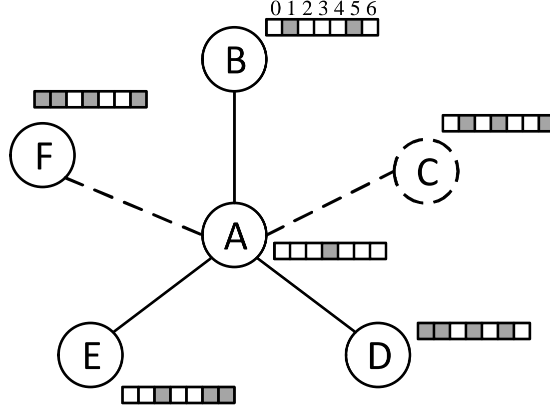

Figure 4.1. An illustration for the Local View Scenario.

And let S_x denote the slice set assigned to download from peer x ($x \in \{B, C, D\}$). Then we need to find a slice downloading assignment so as to minimize the downloading completion time for the given piece:

$$t_{total} = \max_{x \in \{B,C,D\}} \left(\frac{s \cdot |S_x|}{up(x)}\right) ,$$

subjecting to the following constraints:

(1) $\left|\bigcup_{x \in \{B,C,D\}} S_x\right| = m$,

(2) $\forall x_1, x_2 \in \{B, C, D\}$, if $x_1 \neq x_2$, then $S_{x_1} \cap S_{x_2} = \varnothing$.

It is easy to figure out that if $up(x)$ ($x \in \{B, C, D\}$) is constant, one optimal assignment is to assign slices to B, C and D evenly according to their upload rates, i.e., for peer x ($x \in \{B, C, D\}$), we request $\dfrac{up(x) \cdot m}{\sum_{y \in \{B,C,D\}} up(y)}$ number of slices from it.

However, in practice, $up(x)$ ($x \in \{B, C, D\}$) can vary dramatically even in a short period. In addition, the peers in the system can also be highly dynamic, leaving or joining the system at their own will. For example, if peer C leaves the system while peer A is still downloading some slices of piece 1 from it, it may take peer A certain amount of time to

be aware of peer C's leaving and then resend requests to peers B and C for those slices that should previously be downloaded from peer C. Similarly, if later peer A connects to peer F who also has piece 1, to exploit F's upload capacity for piece 1, again after requesting some slices from peer F, peer A needs to cancel these slices from peers B and D to avoid duplications, which would also introduce certain amount of time and bandwidth overhead. It is worth noting that the rarest first strategy in BitTorrent may make the situation even worse. In Fig. 4.1, peer A may decide to download pieces 0, 2 or 4 first since they are the rarest in the local view of peer A, even piece 1 can be downloaded much faster. In fact, if piece 1 is downloaded first, when the downloading finishes, pieces 0, 2 and 4 may have more copies in the local view of peer A (since peers B, C, D and E may download these pieces from their other neighbors during peer A downloads piece 1), leading to even faster downloading speed rather than downloading them right now.

On the other hand, if we adopt the RaptorQ coding technique to encode each piece into different symbols from different peers (which will be further explained in Section 5), the overall performance can be automatically maximized even without solving the above optimization problem. For example, if piece 1 is encoded into different symbols at peers B, C and D. Peer A can simply request peers B, C and D to keep generating and sending different symbols of piece 1 until it collects enough symbols to decode the piece and sends the updated piecemap to its neighbors. It is easy to see that the network dynamics (such as bandwidth variations) will not affect peer A, since the number of symbols sent from peers B, C and D will automatically change with the network dynamics. Also, if peer C leaves and peer F becomes a new neighbor, peer A can easily exploit peer F's upload rate by simply requesting F to generate and send different symbols of piece 1. Again, the number of symbols sent from peers B, C (before it leaves), D and F (after it becomes A's neighbor) will be automatically optimized to minimize the downloading completion time.

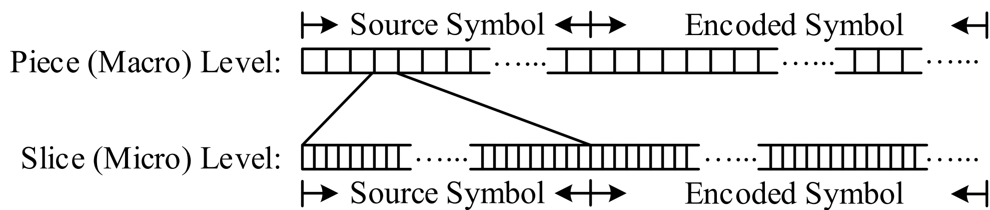

Figure 4.2. An illustration for two level encoding.

4.2 Analysis on Global View Scenario

This scenario further investigates why BitTorrent adopts the rarest first strategy and how we can compensate with RaptorQ coding. Remember that in BitTorrent, the original file is evenly divided into pieces. Given that the peer-to-peer environment is highly dynamic, if one of the pieces is very rare in the system and it happens that all the peers that have this piece suddenly leave the system, other peers thus can never finish the downloading until at least one peer with this piece comes back to the system. To minimize the probability that such situations happen, one optimal solution is to make all the pieces have even number of copies in the system. The corresponding mechanism to achieve this is to always download the rarest piece in the system first. As this mechanism needs global information, which can be very costly in a large-scale peer-to-peer environment, a reasonable approximation is thus to first download the piece that is the rarest one in the local view, i.e., among all the neighbors of a peer.

However, there may be some drawbacks for this approximation. First, a piece is the rarest in the local view may not be the rarest in the global view. More importantly, if we allow a peer can download one piece from multiple neighbors, this mechanism actually slows down the speed for a peer to finish downloading a piece, since the peer chooses the locally rarest one rather than other pieces with more copies locally, losing the opportunities to exploit more peers contributing larger aggregate upload bandwidth to deliver these pieces to the peer faster. In addition, rarest first is also known to cause that when a peer is close to finish downloading, it may take enormous time to locate and download the remained missing pieces.

To address this issue, we also conduct RaptorQ coding at the inter-piece level as shown in Fig. 4.2. Assume the original file can be evenly divided into N pieces. Besides treating a slice as a source symbol and encoding on one piece, we now treat each piece as a source symbol and further encode on the entire file (or each part of the file if the file is too large and needs to be divided into parts) into N source symbols and N repair symbols. For simplicity, here we omit the encoding overhead and assume that collecting any N symbols is enough to decode the original file. Then to alleviate the situation described at the beginning of this subsection, now rarest first changes to that we roughly need to download one of the rarest N symbols first. Since a peer only needs N symbols to decode the original file, when a peer choose one of the rarest N symbols first in the local view, it now have much larger flexibility and can largely ignore the rarest first and select the symbol (piece) that can be provided by more neighbors simultaneously to speed up the downloading. We use a simple example to further illustrate how the inter-piece level RaptorQ coding can help alleviate the situation described at the beginning of this section. Assume a file contains only 2 pieces, which can be further encoded into 4 symbols. There is only one seeder in the system and then two new leechers join the system. By rarest first, each leacher will randomly choose one piece (or equivalently, symbol) to download. Then after each leacher downloads one symbol, the seeder leaves the system. In this case, it is easy to calculate that the probability that two leachers happen to download the same symbol and thus there are not enough symbols in the system to finish the downloading. For BitTorrent, the probability is $2 \times (\frac{1}{2})^2 = \frac{1}{2}$, while by RaptorQ coding, the probability becomes $4 \times (\frac{1}{4})^2 = \frac{1}{4}$, which is much less than that of BitTorrent. In next chapter, we will present the detailed design of our RaptorQP2P protocol to maximize the performance for P2P file distribution by RaptorQ coding.

CHAPTER 5

RAPTORQP2P PROTOCOL DESIGN

For ease of exposition, we borrow some terminology from BitTorrent to help elaborate our protocol design. In RaptorQP2P, we set each piece length as $1600KB$, which can be further divided into 100 source symbols. The size of each source symbol is thus $16KB$. Recall that in BitTorrent, one peer should download one piece from only one peer unless that peer leaves the swarm or gets choked. Also, an unfinished piece cannot be shared to other peers until its downloading has been finished. Different from BitTorrent, by utilizing RaptorQ codes, our protocol allows multiple peers to send the same piece to a peer simultaneously, which can be automatically optimized by the RaptorQ coding as discussed in the previous section. Moreover, to maximize the utilization of peers' upload capacities, our protocol also allows *opportunistic transmissions*, where a peer can send the received encoded symbols of a piece to other peers even if the peer does not have the full piece yet.

As discussed in the previous chapter, one challenge that lies in such a mechanism is that to we must ensure that different neighbors generate and send different encoded symbols so as to avoid duplications, which has the effect of wasting upload capacity. To this end, we exploit the fact that each symbol generated by RaptorQ coding for a given piece has a unique symbol number, and propose an intelligent symbol scheduling algorithm to guarantee that the symbols sent out from one neighbor are different from the symbols sent out from all the other neighbors. The main idea is that since a peer only has a limited number of neighbor slots, we only allow the symbols with the specific symbol numbers to be sent from a given neighbor slot. Specifically, when a peer connects to a new neighbor, it will assign an empty neighbor slot to the neighbor and inform the neighbor its neighbor slot number during

the initial piecemap exchange. If the neighbor has a full piece that the peer does not have, the neighbor will only send those symbols encoded from that piece whose symbol numbers must be equal to the neighbor's slot number after taking *mod* on the number of the peer's neighbor slots. Since different neighbors of a peer have different neighbor slot numbers, we can guarantee that the encoded symbols we receive from one neighbor are different from those received from another neighbor. In addition, to deal with peer churns, when requesting a piece from a neighbor, we also let a peer include the maximum symbol number that it has received so far for the piece, so that if a neighbor leaves after sending some encoded symbols, the new neighbor can continue sending later symbols starting from the maximum symbol number indicated during the piece request.

The pseudo code of the algorithm is summarized in Fig. 5.1, where $NSize$ is the number of neighbor slots, i is the sender's neighbor slot number assigned by the receiver. We use $maxsymbol[j]$ to denote the maximum symbol that the receiver has received for piece j. This pseudo code can divide into two parts. The first part (line 1-15) deals with the case that the sender has the full piece j and the second part (line 16-25) handles the case that the sender only has the partial data of piece j. If the sender has a full piece, it will send the symbols depending on its neighbor slot number i assigned by the receiver. In particular, the sender will first check whether $(maxsymbol[j] - i)$ can be exactly divided by $NSize$. If so, it will send the symbol with the number of $maxsymbol[j] + NSize$ to the receiver. Otherwise, the sender will calculate the first symbol number that is greater than the current $maxsymbol[j]$ and can be exactly divided by i, and then send the symbol to the receiver. Moreover, even the sender only has part of piece j, it can check its received symbols in *symbolmap* and find the symbol whose number is greater than the $maxsymbol[j]$ and can be exactly divided by $NSize$ after minus i. If so, the sender can still "opportunistically" send the symbol to the receiver, so as to better utilize its upload capacity.

This mechanism can make sure that all the neighbors can send one piece to one receiver simultaneously and even one peer have not finished downloading one piece, he can

```
Algorithm IntelligentSymbolScheduling(piece j)
1:  if Sender has the full piece j,
2:     if (maxsymbol[j] − i)%NSize == 0,
3:        maxsymbol[j] = maxsymbol[j] + NSize;
4:        Generate and send the symbol with the
5:           number maxsymbol[j];
6:     else
7:        for a = 0..NSize − 1,
8:           maxsymbol[j] + +;
9:           if (maxsymbol[j] − i)%NSize == 0,
10:             break;
11:          end if
12:       end for
13:       Generate and send the symbol with the
14:          number maxsymbol[j];
15:    end if
16: else if Sender only has the partial piece j,
17:    for b = 0..receivedsymbols,
18:       if symbolmap[j][b] > maxsymbol[j] and (symbolmap[j][b] − i)%NSize == 0,
19:          maxsymbol[j] = symbol[j][b];
20:          Send the symbol with the number
21:             maxsymbol[j];
22:          break;
23:       end if
24:    end for
25: end if
```

Figure 5.1. The algorithm for intelligent symbol scheduling.

still send the received encoded symbols to others and no duplicated symbol would be send. Now recall Fig. 4.1, let's say peer F is neighbor 0 of peer A, peer B is neighbor 1 of peer A, peer C is neighbor 2 of peer A, peer D is neighbor 3 of peer A, peer E is neighbor 4 of peer A. Now assume peer A does not have any symbol of piece 1 he will ask all the neighbors to send piece 1 to him. Due to peer F is the neighbor 0 of peer A, so he would send symbol 5, 10, etc., peer B would send symbol 6, 11, etc., peer C would send symbol 7, 12, peer D would send symbol 8, 13 peer E should send 9, 14, but due to he does not have a full piece of piece 1, he will check peer $A'smaxsymbol[1]$ (the maximum symbol of piece 1) if peer E has the symbol number which larger than peer $A'smaxsymbol[1]$ and the symbol number

($symbolnumber$ - 4) mod 5 is equal to 0 (here the number 4 is the position of peer E in peer A's neighbor, 5 is the $NSize$), then peer E will send the symbol and peer A will save this in its $symbolmap[j][n]$. For example, if peer E has symbol of $symbol9$, $symbol10$ and $symbol17$, only symbol9 can be sent to peer A, since $(9-4)\%5 = 0$. If later, peer F leaves the swarm, peer B, C, D, E can still send peer A a certain number of symbols to let peer A decodes piece 1. If peer G joins to the swarm after peer F leaves, G becomes new neighbor 0 of peer A and peer G also has piece 1, peer A will tell peer G peer $A'smaxsymbol[1]$ and peer G' neighbor slot of peer A is 0, then peer G can send symbols to peer A. For example, if peer G find peer $A'smaxsymbol[1]$ is 25, then he check his neighbor slot of peer A's neighbor is 0 and he will send symbol 30, 35, etc.; if peer G find peer $A'smaxsymbol[1]$ is 28, then he check his neighbor slot of peer A's neighbor is 0, do a calculation to get the nearest symbol number to $symbol28$ which should correlative to G' neighbor slot, then he will find symbol 30 is suitable to send to peer A.

In the RaptorP2P protocol, we also adopt the inter-piece level encoding discussed in the previous chapter. In particular, each seeder in the system will encode the file at the piece level so that for a file with N pieces, we now have $2N$ piece level symbols for downloading. Then when a peer selects a piece to download from its neighbors, it can choose among the rarest N pieces instead of the rarest piece in the BitTorrent protocol. In next chapter, we will conduct extensive simulations with the real world traces measured from the BitTorrent system to evaluate our RaptorQP2P protocol, which further demonstrate the effectiveness of our RaptorQ based protocol design.

CHAPTER 6

PERFORMANCE EVALUATION

6.1 Methodology

We run extensive simulations to evaluate RaptorQP2P. To make it more practical, our simulation is driven by the traces measured from the real BitTorrent system from Wang et al. (2012b), which mainly contain the peer online/offline patterns in a swarm with various real world network dynamics such as peer churns and flashcrowds. The upload capacity of each peer is randomly chosen between $240KB/s$ to $720KB/s$. The default peer number is set to 1000 and the file size is $512MB$. Other configurations are adopted from the typical settings used in other authors Wang et al. (2012b), Magnetto et al. (2010), Piatek et al. (2007) and Wang et al. (2011). For comparison, we also implement the BitTorrent protocol and focus on the downloading completion time which is the time that a peer has downloaded enough pieces (or piece level symbols) to get the original file. In addition, we also vary the peer number from 200 to 1000 and the file size from $32MB$ to $512MB$ to investigate the scalability of our solution.

Our simulator is based on time-driven. Each time instance all the peers would first maintain their neighbors, then put requests to their neighbors and send the data to their neighbors. In the stage of neighbor maintenance, each peer should maintain three layers. The first layer is member maintenance. In this layer, tracker responses a list of peers who sharing the same file. The second layer is general neighbor maintenance, where a peer can select a list of peers from the tracker's response and connect to them as general neighbor to each other. The last layer is mesh neighbor maintenance. In this layer, a peer will select some peers who have his interesting pieces from its general neighbors, and add those peers to

its mesh neighbor. So the real data transmission only happens among the mesh neighbors. Also, the tit-for-tat mechanism is used in this stage. The second stage is putting requests to those peers who have his interesting piece. In BitTorrent, for a given piece, a peer only puts request to those mesh neighbors who have the whole piece. However, in RaptorQP2P, a peer can put request to those mesh neighbors who have any encoded symbols for that piece and tell them the maximum symbol number for that piece. The peer will also inform all mesh neighbors their corresponding neighbor slots. The last stage is sending data to peer's neighbors. In this stage, BitTorrent will use the strict priority, rarest first, random first piece and endgame mode. In our RaptorQP2P, peers use the mechanisms described in Chapter 5.

6.2 Performance Results

Fig. 6.1 shows the results of the total downloading completion time (i.e., the time for all the peers get the original file) as a function of file size. As expected, with the file size becoming large, the total downloading completion time of both RaptorQP2P and BitTorrent increases linearly. However, the total downloading completion time of RaptorQP2P increases much slower than that of BitTorrent. This is because as the file size increases, our piece level encoding can make the piece selection process more flexible, which allows a peer to choose a piece with more copies among its neighbors and further speed up the time of downloading a piece by our protocol. Moreover, when the file size increases to $512MB$, the time taken by RaptorQP2P is only 58.6% of the time taken by BitTorrent, which is roughly a performance gain of 41.4%. This further demonstrates the effectiveness of our RaptorQ based protocol on distributing large files, which can be of great demands in nowadays applications such as high-definition medical or satellite image distribution among different sites and virtual machine image distribution among different cloud servers, where a typical file size can be hundreds of megabytes or even more.

To better understand the performance of each individual peer, we also investigate the CDF of each peer's downloading completion time with 1000 peers and a $512MB$ file,

which is shown in Fig. 6.2. It is easy to see that all the peers using RaptorQP2P have much smaller downloading completion time than those using BitTorrent. Besides, since in the real environment, peers may join the system at different time and their online/offline patterns can be totally different, this explains why the CDF curve of RaptorQP2P expands between 1144s and 1401s. However, no matter when to join the system, all the peers using BitTorrent finish the downloading at the time closer to each other (roughly around 2339s). This is because under the network dynamics and peer churns, the well-known last piece problem of BitTorrent (i.e., it is often very hard for a peer to identify the last missing piece for downloading) becomes more severe, as those peers with the last missing piece that a peer is looking for may dynamically leave the system and then rejoin later. On the other hand, due to the piece level encoding, the peers using RaptorQP2P have much flexible choices for their last missing pieces, since for a N-piece file, we now have roughly N options for the last missing piece.

We next examine how our solution scales with different number of peers. The results are shown in Fig. 6.3. When the number of peers changes from 200 to 1000, the total downloading completion time of BitTorrent gradually increases with some small fluctuations. This is because the peer traces are collected from the real BitTorrent system, where with more peers joining/leaving in the system, more peer churns and flashcrowds may happen, not only degrading the overall performance but also bringing more fluctuations. RaptorQP2P, however, has much stable performance when the number of peers changes. This is because RaptorQ coding can automatically help optimize the symbol downloading among different neighbors even they may come and leave at any time. Moreover, our opportunistic transmission scheme can further exploit a peer's upload rate and speed up the file distribution process even when a piece is not fully downloaded at the peer. To this end, we take a closer look at how much data traffic is delivered through our opportunistic transmission scheme and the results are shown in Fig. 6.4. It is easy to see that the portion of the traffic delivered by opportunistic transmissions (denoted as opportunistic traffic) increases steadily with the

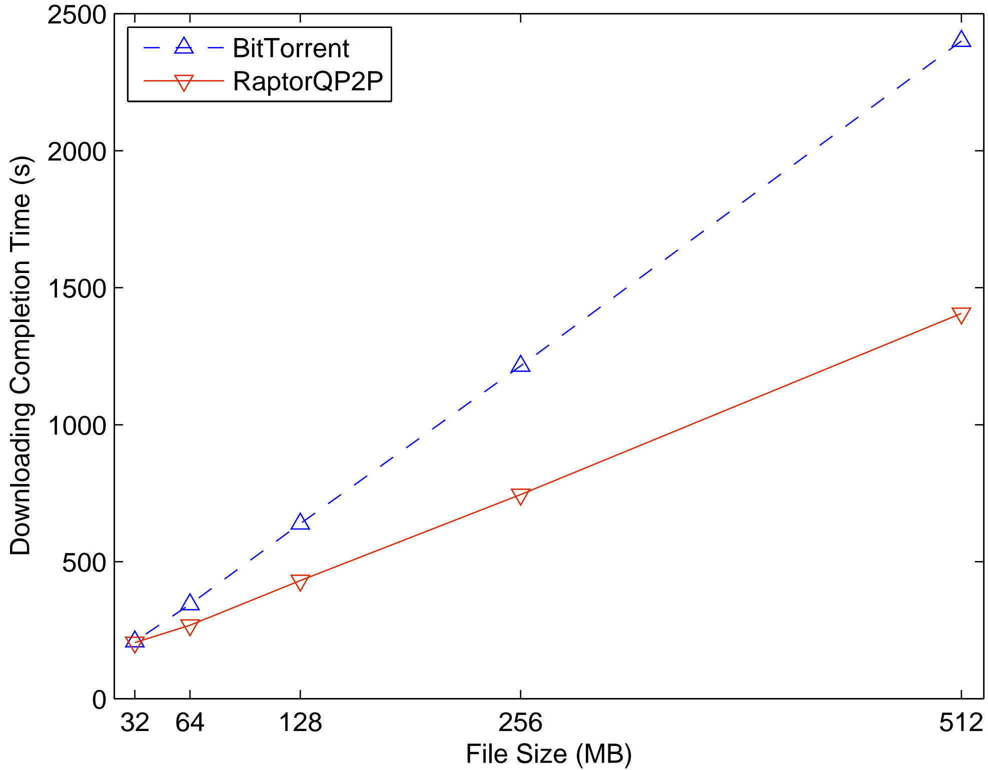

Figure 6.1. Total downloading completion time as a function of file size for 1000 peers.

file size (up to 39.3% of the total traffic when the file size is $512MB$), since larger files have more pieces and thus bring more chances for opportunistic transmissions, which also explains why RaptorQP2P performs much better than BitTorrent especially when dealing with large files.

In practice, it is often very important to understand how long for a individual peer to receive the first piece when flash crowds happen. In Fig. 6.5, we simulate a flash crowds scenario where 1000 peers join a swarm simultaneously and download a $512MB$ size file. The results show that comparing to BitTorrent protocol, our RaptorQP2P can efficiently resist the flash crowd influence and help the peers receive the first piece quickly, i.e. , all the peers can receive the first piece in 60 seconds by RaptorQP2P. However in BitTorrent protocol, the slowest peer cannot receive the first full piece until 124 seconds. The time taken by RaptorQP2P is only 48.4% of the time taken by BitTorrent, which is roughly a performance

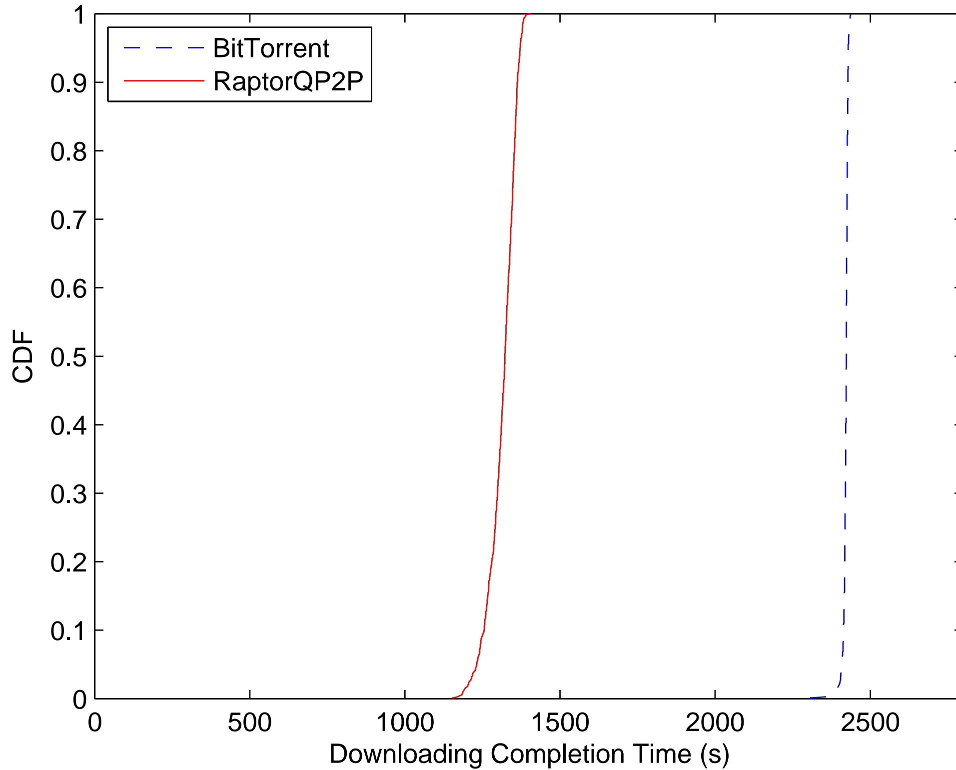

Figure 6.2. CDF of individual peer's downloading completion time for a 512 MB file and 1000 peers.

gain of 51.6%. This further demonstrates the effectiveness of our RaptorQ based protocol on resisting the impacts caused by flash crowds.

Another practical issue is that when downloading a file by P2P, users often become impatient if they wait the first data for a long time, thinking this swarm may not be active and then leaving the swarm. Our RaptorQP2P protocol can significantly reduce the time to wait for the first data. As shown in fig. 6.6, we measure the time of each peer getting the first data with 1000 peers and a 512MB file. It is easy to see that all the peers using RaptorQP2P have much smaller time to getting the first data than those using BitTorrent. We can see from this CDF, in RaptorQP2P, more than 95% of peers can get the first data before 35 seconds, and all the peers can get the first data in 54 seconds. But in the BitTorrent, more than 95% peers can get the first data in 75 seconds, and all the peers can get the first data in

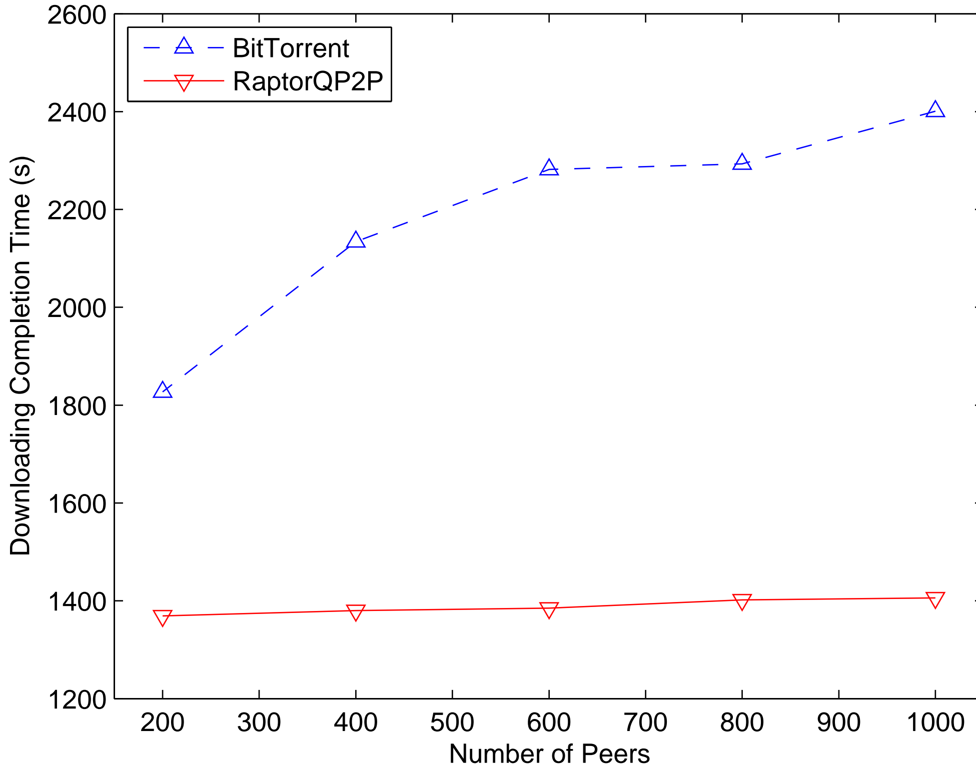

Figure 6.3. Total downloading completion time as a function of the number of peers for a 512 MB file.

85 seconds. Since in our protocol, the peers can send the encoded symbols to his neighbors though they do not finish downloading the piece, so all peers in RaptorQP2P have much more opportunities to get the first data than in BitTorrent, which explains why the CDF shows RaptorQP2P has a smaller time to getting the first data.

We next examine how the average time of getting the first data scales with different number of peers. The results are shown in Fig. 6.7. When the number of peers changes from 200 to 1000, the average time of getting the first data by BitTorrent gradually decreases with some small fluctuations where the fluctuations are mainly caused by peer churns and flash crowds. But in the RaptorQP2P, comparing to the BitTorrent, the average time of getting first data stays very low and is more stable with the peer number increasing. This is because the opportunistic transmission scheme in RaptorQP2P can better exploit a peer's upload

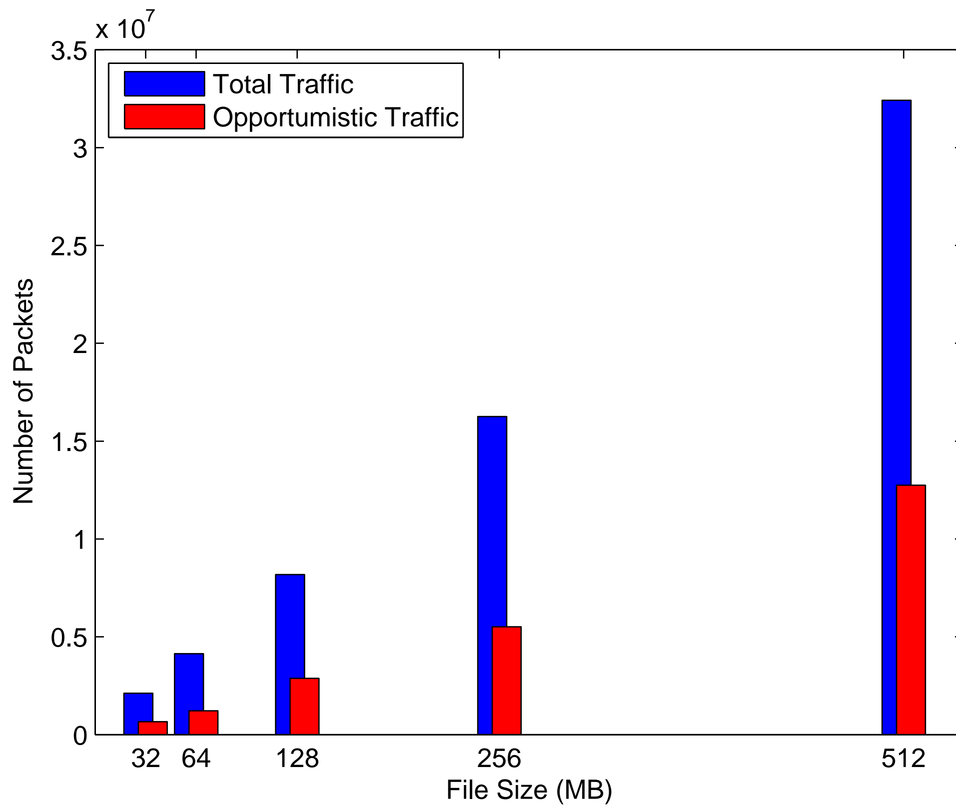

Figure 6.4. The contribution of opportunistic transmissions to the whole system as a function of file size with 1000 peers.

capacity and speed up the file distribution process even when a piece is not fully downloaded at the peer, so that each peer has more chances to get the first data.

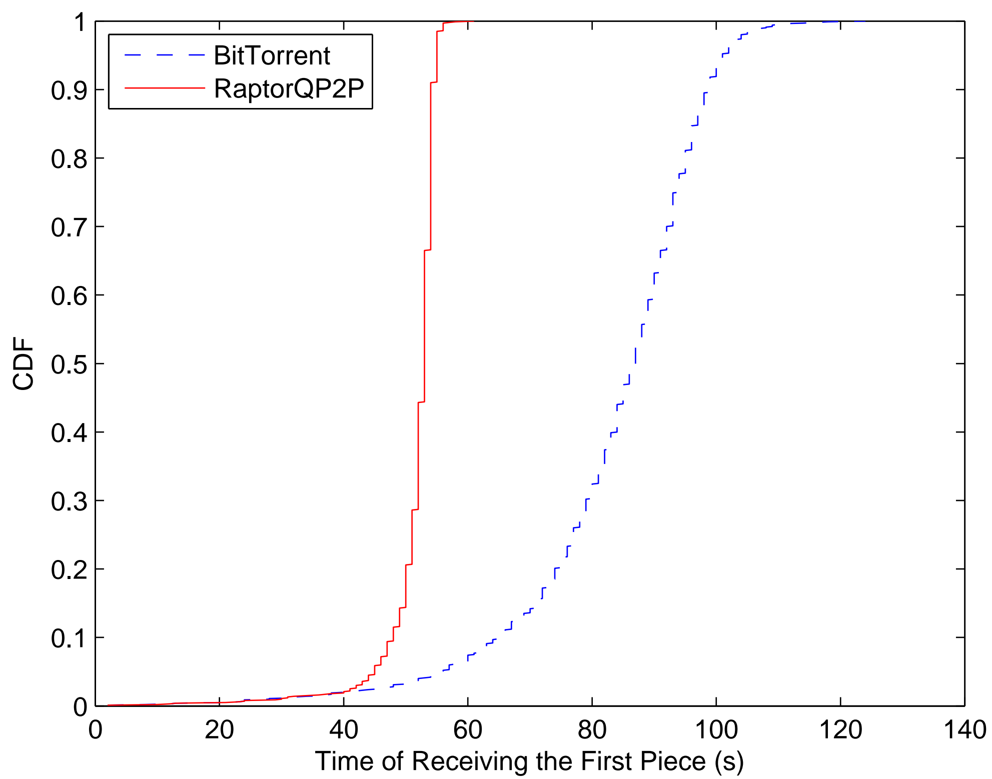

Figure 6.5. CDF of individual peer's getting first piece in a flash crowds condition for a 512 MB file and 1000 peers.

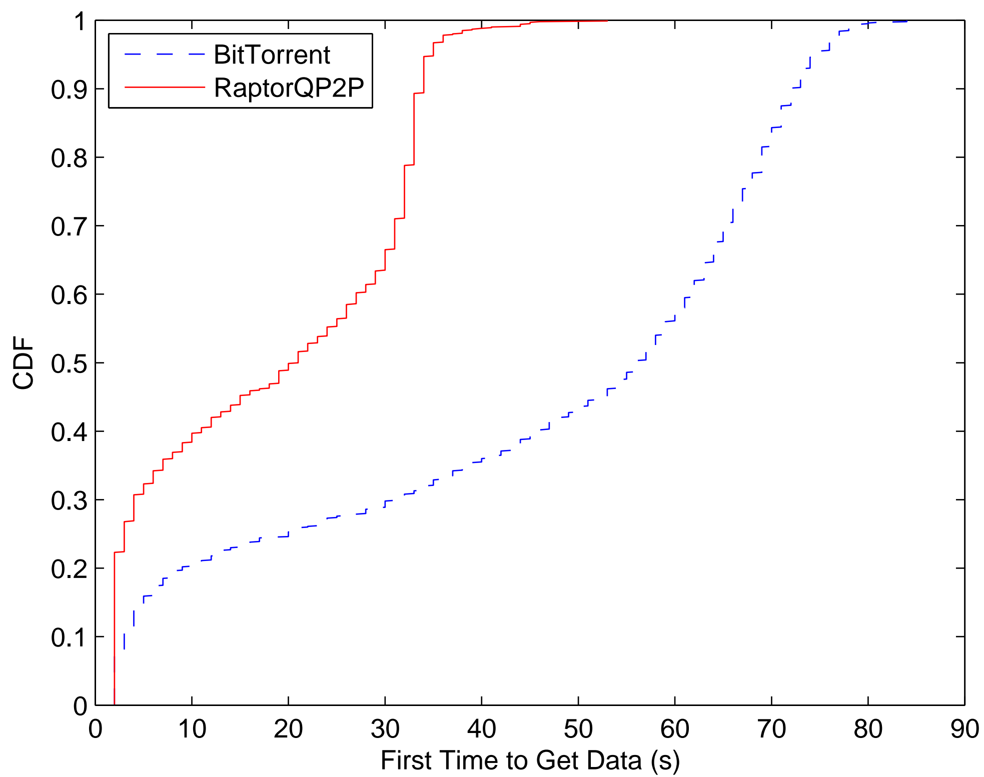

Figure 6.6. CDF of individual peer's getting first data for a 512 MB file and 1000 peers.

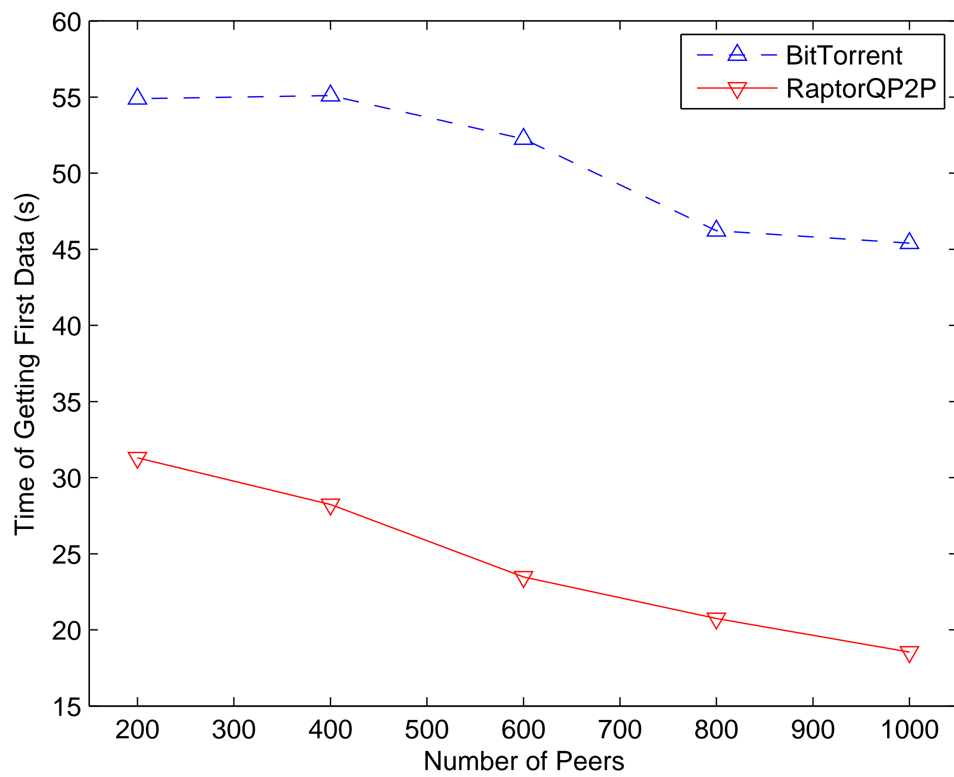

Figure 6.7. Average time of getting first data as a function of the number of peers for a 512 MB file.

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis, we presented a novel RaptorQP2P protocol which applies the RaptorQ coding technique into P2P file distribution, where the design was motivated by our in-depth study on the limitations of BitTorrent under various dynamic network environments. We first encode the piece into symbols and utilizing the feature that RaptorQ codes can generate rateless identical symbols to fully utilize each peer's upload capacity and avoid the influence by network dynamic, peer churn and flash crowds. Then we take a global view to see the whole file. To decrease the possibility of missing piece happened, we encode all the pieces to generate the encoded pieces and when the receiver receiving a certain number of encoded pieces, then he can decode the whole file. This piece level encoded mechanism can increase the piece variety and resist the influence by peers who have a certain piece leave the swarm suddenly which will caused a missing piece happened. We conducted extensive simulations driven by the real traces measured from the BitTorrent system. The results showed that our protocol can scale well with different user populations and achieve much better performance than BitTorrent, especially when dealing with large files.

Besides further evaluating our protocol in a prototype system, we are also interested in investigating how our protocol would perform under other type of networks such as cloud and wireless networks.

Another interesting idea is to apply online social networks (OSNs) to P2P file sharing. In most cases, people who are friends in real world are friends in OSNs . This provides a better incentive for users to share more resources in P2P file distribution and keep staying in the system to upload after finishing download. One preliminary work has been done by Su

and Wang (2013). It is thus interesting to further explore along this direction and extend our protocol there.

BIBLIOGRAPHY

BIBLIOGRAPHY

Bharambe, A. R., C. Herley, and V. N. Padmanabhan (2006), Analyzing and Improving a BitTorrent Networks Performance Mechanisms, in *INFOCOM*.

BitTorrent (2001), http://www.bittorrent.com.

Bouras, C., N. Kanakis, V. Kokkinos, , and A. Papazois (2013), Embracing RaptorQ FEC in 3GPP Multicast Service, *Wireless Networks*, 19(5), 1023–1035.

Cohen, B. (2003), Incentives Build Robustness in BitTorrent, in *IPTPS*.

Eittenberger, P. M., T. Mladenov, and U. R. Krieger (2012), Raptor Codes for P2P Streaming, in *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*.

Legout, A., G. Urvoy-Keller, and P. Michiardi (2006), Rarest First and Choke Algorithms Are Enough, in *ACM IMC*.

Liu, J., S. G. Rao, B. Li, and H. Zhang (2008), Opportunities and Challenges of Peer-to-Peer Internet Video Broadcast, *Proceedings of the IEEE*, 96(1), 11–24.

Luby, M. (2002), LT Codes, in *IEEE Symposium on Foundations of Computer Science*.

Luby, M. (2012), Best practices for mobile broadcast delivery and playback of multimedia content, in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*.

Luby, M., A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder (2011), RaptorQ Forward Error Correction Scheme for Object Delivery, in *IETF*.

MacKay, J. (2005), Fountain Codes, *IEE proceedings-Communications*, 152(6), 1062–1068.

Magnetto, A., S. Spoto, R. Gaeta, M. Grangetto, and M. Sereno (2010), Fountains vs Torrents: the P2P ToroVerde Protocol, in *IEEE/ACM Annual International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*.

Miguel, B., C. Toh, C. T. Calafate, J.-C. Cano, and P. Manzoni (2013), RCDP: Raptor-based content delivery protocol for unicast communication in wireless networks for ITS, *Wireless Networks*, 15(2), 198–206.

Milojicic, D. S., V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu (2002), Peer-to-Peer Computing, *Tech. rep.*, HP Labs.

Piatek, M., T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani (2007), Do incentives build robustness in BitTorrent?, in *USENIX NSDI*.

Shokrollahi, A., and M. Luby (2011), Raptor Codes, *Foundations and Trends® in Communications and Information Theory*, 6(3-4), 213–322.

Spoto, S., R. Gaeta, M. Grangetto, and M. Sereno. (2010), BitTorrent and Fountain Codes: Friends or Foes ?, in *International Workshop on Hot Topics in Peer-to-Peer Systems (HotP2P)*.

Su, Z., and F. Wang (2013), Enhancing Bit Torrent with Twitter-like Social Relationships, in *MAESC*.

Wang, F., J. Liu, and M. Chen (2012a), Cloud-Assisted Live Media Streaming for Globalized Demands with Time/Region Diversities, in *INFOCOM*.

Wang, H., F. Wang, and J. Liu (2011), On Long-Term Social Relationships in Peer-to-Peer Systems, in *IEEE/ACM IWQoS*.

Wang, H., F. Wang, and J. Liu (2012b), Accelerating Peer-to-Peer File Sharing with Social Relations: Potentials and Challenges, in *INFOCOM*.

Wu, Y., C. Wu, B. Li, X. Qiu, and F. C. Lau (2011), CloudMedia: When Cloud on Demand Meets Video on Demand, in *the Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS)*.

VITA

Zeyang Su

Zeyang Su is a master student in computer science and information department of University of Mississippi. He received the Bachelor Degree in China University of Geoscience (Beijing).