RaptorQP2P: Maximize the Performance of P2P File Distribution with RaptorQ Coding

Zeyang Su* Feng Wang* John Daigle* Haiyang Wang† Tong Shan*
*University of Mississippi, University, MS, USA †University of Minnesota at Duluth Duluth, MN, USA

***Abstract*—Presented herein is a new protocol, RaptorQP2P, for reliable peer-to-peer sharing of large files–currently on the order of hundreds of MB but expected to grow to the terabyte range–over networks. 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. At the lower layer, each source/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 block have been received, whether source or repair, the block can be reconstructed. Similarly, the file can be reconstructed using a sufficient number of arbitrary distinct blocks. With real world traces measured on the BitTorrent system, we have conducted extensive simulations to evaluate and compare the performance of RaptorQP2P and BitTorrent. The results indicate that the new protocol handles network dynamics and peer churn smoothly, has excellent scalability with respect to both file size and user population, and performs well. As an example, for a 512 MB file distributed to 1000 peers, download completion time using the new protocol was found to be approximately 45.5% of the completion time required using BitTorrent.**

I. Introduction

The objective of this paper is to demonstrate the application of fountain coding for reliable peer-to-peer sharing of large files over networks. Presented herein is the protocol RaptorQP2P, which can provide reliable delivery of files exceeding hundreds of MB–such as those that occur or will soon arise in virtual machine image migration in cloud computing, big data processing, and ultra high-definition video–to thousands of peers. The new protocol features two levels of RaptorQ [7] encoding. 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 of 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, equal to or slightly exceeding the number of source blocks of the file have been received, the receiver is able to reconstruct the entire file.

In addition to the 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 [2]. 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. Indeed, in the performance analysis comparison presented here, blocks and pieces have the same size as do symbols and slices.

RaptorQ coding is the most advanced technology in the series of practical implementations of fountain coding techniques. As will be explained in Section II-A, in the standard RaptorQ encoding process, a file is partitioned into a collection of source blocks, and each of these source blocks is 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, equal to or 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 a number of symbols equal to the number of source symbols provides a decoding success probability of 99%, and receiving two symbols more than the number of source symbols provides a decoding success probability of 99.9999%. When all source blocks have been received, the entire file can be reconstructed.

A brief review of the BitTorrent protocol is given in Section II-B. BitTorrent's [2] 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 includes four strategies [4]: strict priority, rarest first, random first piece and endgame mode. Recent studies [6] show that these mechanisms can help BitTorrent achieve near optimal performance when the network is under steady state. However, it is also reported 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 [11][15].

An in-depth analysis of BitTorrent is presented in Section III. The analysis considers limitations of BitTorrent in highly dynamic network environments. By investigating both local and global view scenarios, it is clearly identified how network dynamics and user churn can significantly degrade performance. At the end of the discussions on each of scenarios in Section III, it is shown how the properties of RaptorQ can be used to great advantage in overcoming these limitations.

Design of the new RaptorQ-based protocol is described in Section IV. It will be seen that by using RaptorQ, a peer can receive a certain piece from all the neighbors who have that piece and, in addition, the peers that have only slices can send those along. Use of RaptorQ naturally increases the variety of pieces in the system so even if some peers having the rarest pieces leave the system, other peers still can provide

978-1-4673-6432-4/15/$31.00 ©2015 IEEE

pieces to complete the download. This 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. We integrate these novel designs and develop a new protocol named RaptorQP2P.

In Section V, we report the results of extensive simulations of both RaptorQP2P and BitTorrent with file size and the number of participating peers as parameters. Real-world traces measured on the BitTorrent system were used to drive the simulations and parameter settings were chosen to yield fair comparisons. The results indicate that the new protocol handles network dynamics and peer churn smoothly, has excellent scalability with respect to both file size and user population, and that use of the new protocol can significantly reduce download times over those required for BitTorrent, especially in the case of large files distributed to a large number of peers. E.g., for a 512 MB file distributed to 1000 peers, download completion time using the new protocol is approximately 45.5% of the completion time required using BitTorrent.

A brief overview of our plans to develop a working beta version implementation of the protocol presented here is included in Section VII.

II. Preliminaries

A. RaptorQ Coding

RaptorQ is a recent addition to the family of Raptor codes [14], which are practical codes in the class of fountain codes [10]. Fountain codes, in turn, are rateless codes; that is, they can generate a potentially infinite number of encoded symbols from a finite set of source symbols. Luby transform (LT) codes [9] are the first generation of practical fountain codes. In LT coding, a file is divided into a collection of blocks and each block is further divided into a collection of source symbols. Encoded symbols are then generated by XORing random numbers of source symbols in a well organized way. Ideally, the number of generated encoded symbols can be limitless. The encoded symbols are transferred over the network and collected at the destination, and when a sufficient number of source symbols have been received, the receiver is able to recover the source symbols of each block. Since the encoded symbols are generated randomly, only the number of symbols received at each receiver is important; the specific encoded symbols do not matter.

Different from the LT codes, in RaptorQ, after we divide the file into blocks and symbols, there is a pre-coding stage in which the source symbols are erasure encoded to yield a collection of intermediate symbols, the number of which is slightly larger than the number of source symbols. These intermediate symbols are LT-encoded into an arbitrarily large stream of encoded symbols and transferred over the network. When a destination has received slightly more than the original number of source symbols, typically denoted by k, the intermediate symbols are recovered from the encoded symbols. The original source symbols are then recovered from the intermediate symbols. Again, the required number of encoded symbols received at each destination need be only slightly larger than the number of source symbols. For example if the number of source symbols is $k = 100$, collecting an arbitrary set of 102 encoded symbols will result in successful block recovery with very high probability. In addition to its very low decoding overhead, RaptorQ achieves both encoding and decoding in linear time with respect to file size.

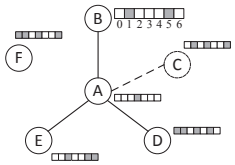

Fig. 1: An illustration for the local view scenario.

B. BitTorrent

BitTorrent [2] is the mostly used leading P2P file sharing system over the Internet. It uses two mechanisms for piece requesting and downloading. The first is *tit-for-tat*, mainly designed to avoid *freeriders* 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. The second mechanism is for piece selection and includes four parts: strict priority, rarest first, random first piece and endgame mode, where the analysis on the former two parts mainly motivates our RaptorQP2P protocol design. Strict priority requires that all the slices in a piece should be downloaded from only one peer, unless that peer leaves the system or is choked. Then a second peer will be selected to download the remained slices. Also, a peer cannot share the slices of a partially downloaded piece with others. 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 means that 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.

III. Analysis on BitTorrent and Its Limitations

In this section, 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 limitations, which further motivates our RaptorQP2P protocol design proposed in next section.

A. Analysis of Local View Scenario

The local view scenario refers to a peer's view of its neighbors. Fig. 1 depicts the situation where peer A is neighboring with peers B, C, D and E. Suppose that peer A needs piece 1, which peers B, C and D have. Under strict priority, peer A must choose only one peer among B, C and D for downloading piece 1. As time progresses and network dynamics change, peer A must adhere to its original choice

even if that choice turns out not to be a good one. For example, if A's original choice is B, C leaves, and F becomes a new neighbor of A, then A cannot switch to F to download piece 1 no matter how high F's upload rate is.

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. 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 $\mathrm{up}(x)$ 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. 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 to minimize the downloading completion time for the given piece:

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

subject 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 see that if $\mathrm{up}(x), x \in \{B, C, D\}$, is constant, one optimal approach is to assign slices to B, C and D proportional to their upload rates; for peer $x \in \{B, C, D\}$, we request $\dfrac{\mathrm{up}(x) \cdot m}{\sum_{y \in \{B,C,D\}} \mathrm{up}(y)}$ slices from x. However, in practice, $\mathrm{up}(x), x \in \{B, C, D\}$, can vary dramatically even over a short period. In addition, the peers leave and join the system at their own will. For example, if peer A is downloading some slices of piece 1 from peer C when peer C leaves the system, then peer A will incur time and communication overhead to arrange to receive those same slices from peers B and D. Similarly, if later peer A connects to peer F, which also has piece 1, to avoid duplication peer A would have to cancel downloads of slices from peers B and D in order to take advantage of F's higher upload capacity for piece 1.

It is worth noting that the rarest first strategy in BitTorrent may make the situation even worse. In Fig. 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 if piece 1 can be downloaded much faster. In practice, if piece 1 were downloaded first, it is possible that peers B, C, D and E would have downloaded pieces 0, 2 and 4 during A's download of piece 1 so that A downloading piece 1 first would have resulted in a faster download.

As will be further explained in Section IV, adoption of RaptorQ to encode each piece into different symbols from different peers facilitates minimization of download time without solving mathematical optimization problems. 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) can all be used to minimize the downloading completion time.

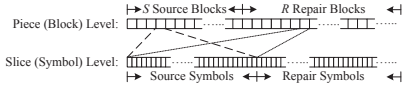

Fig. 2: An illustration for two level encoding.

B. Analysis of Global View Scenario

This section explains why the rarest first strategy exists in the BitTorrecnt protocol but is not strictly needed in RaptorQ. 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 rare and it happens that the few peers that have this piece suddenly leave the system, the other peers will be unable to finish the downloading the file until at least one peer with this rare piece returns to the system. To minimize the probability that such situations occur, one approach is to try to maintain an equal number of copies of each piece in the system. The corresponding mechanism to achieve this is for each peer to always download the rarest piece in the system first. In a large-scale peer-to-peer environment, the global information needed to implement this strategy is costly, and thus a reasonable approximation is to first download the piece that is the rarest among all the neighbors of a peer.

There are some drawbacks to this local approximation. First, a piece that is rarest in the local view may not be the rarest in the global view. More importantly, if a peer could download a single piece from multiple neighbors, the rarest first strategy would reduce the overall download speed because the peer loses the opportunity to exploit the upload bandwidth of more peers. In addition, it is well-known that, in practice, the rarest first strategy sometimes leads to a situation where locating and downloading the last remaining pieces is very time-consuming.

To address the rarity issue, we RaptorQ encode at the file level in addition to the piece level as shown in Fig. 2. First, the original file is evenly divided into S source pieces. These S source pieces are RaptorQ-encoded into S source blocks and R repair blocks. Each source and repair block is further independently RaptorQ encoded into symbols and distributed over the network. Even for large S, whenever any peer collects any S of these source and repair pieces, that peer will be able to recover the entire file with at least 99% probability. Collection of any set of $S + 2$ of the pieces provides virtually 100% recovery. Since any set of $S + 2$ source and repair pieces is equally useful, a peer has much greater flexibility (e.g., if $R = S$, then choosing the rarest first roughly becomes choosing the rarest S first, which can be largely ignored) so as to select the piece that provides the highest download speed.

A simple example further illustrates how file level RaptorQ

encoding alleviates the rarest piece problem. Assume a file is split into 2 pieces, which are RaptorQ encoded into 2 source and 3 repair blocks ($S = 2$, $R = 3$). There is only one seeder in the system and then two leechers join the system. By rarest first, each leecher will randomly choose one piece to download. Then after each leecher downloads one piece, the seeder leaves the system. In this case, the probability that two leechers happen to download the same piece in the BitTorrent case is 0.5 while in the RaptorQ case it is 0.2. Thus, the probability that downloading can be completed by all peers is 0.5 for the BitTorrent case and 0.8 for the RaptorQ case.

IV. RaptorQP2P Protocol Design

We turn now to the detailed design of our RaptorQP2P protocol. 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 with other peers until its downloading has been finished. Different from BitTorrent, by utilizing RaptorQ encoding, our protocol allows multiple peers to send encoded symbols of the same piece to a peer simultaneously. 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 section, one challenge that lies in such a mechanism is that we must ensure that different neighbors generate and send different encoded symbols to avoid duplication, 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 ID, and propose an intelligent symbol scheduling algorithm that guarantees 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 A only has a limited number of neighbor slots, we number the slots 0 to $N-1$, where N is the allowable number of neighbors. When peer A connects to a new neighbor, say B, it assigns an empty slot to the neighbor and inform the neighbor its neighbor slot number, say n, during the initial piecemap exchange. If peer B has a full piece that peer A does not have, peer A may request that piece from peer B and B is then allowed to send generated symbols of that piece to peer A only when the symbol ID of a generated symbol fulfills $\text{symbol ID} \mod N = n$. 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 any other neighbor. In addition, to deal with peer churns, when requesting a piece from a neighbor, we also include the maximum symbol ID received so far for the piece. Thus, if a neighbor leaves after sending some encoded symbols, the new neighbor can continue sending later symbols starting from the maximum symbol ID indicated during the piece request.

The pseudocode of the algorithm is summarized in Fig. 3, where N is the number of neighbor slots and i is the sender's neighbor slot number assigned by the receiver. We

Algorithm Symbol Scheduling (piece j)

```
1:  if Sender has the full piece j,
2:      maxsymbol[j]++;
3:      while (maxsymbol[j])%N != i,
4:          maxsymbol[j]++;
5:      end while
6:      Generate and send the symbol with the ID
7:        equal to maxsymbol[j];
8:  else if Sender only has the partial piece j,
9:      for b = 0..receivedsymbols,
10:       if symbolmap[j][b] > maxsymbol[j] and
            symbolmap[j][b]%N == i,
11:         maxsymbol[j] = symbol[j][b];
12:         Send the symbol with the ID equal to
13:           maxsymbol[j];
14:         break;
15:       end if
16:     end for
17: end if
```

Fig. 3: The algorithm for intelligent symbol scheduling.

use $\text{maxsymbol}[j]$ to denote the maximum symbol ID that the receiver has received for piece j. Lines 1-7 deal with the case that the sender has the full piece j and lines 8-17 handle the case where the sender has only part 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, given current $\text{maxsymbol}[j]$, the sender will increase the value of $\text{maxsymbol}[j]$ to the first symbol ID greater than the current $\text{maxsymbol}[j]$ such that $\text{maxsymbol}[j] \mod N = i$, and then send the symbol with that symbol ID to the receiver. If the sender has only part of piece j, it checks its received symbols for piece j in its $\text{symbolmap}[j]$ and finds the first symbol, say, the b-th symbol $\text{symbolmap}[j][b]$ whose ID is greater than the current $\text{maxsymbol}[j]$ such that $\text{symbolmap}[j][b] \mod N = i$. If such a symbol is found, the sender can *opportunistically* send the symbol to the receiver, which better utilizes its upload capacity.

In the RaptorP2P protocol, we also adopt the file level encoding discussed in the previous section. In particular, each seeder in the system will encode the file so that a file with S pieces will have $S + R$ piece-level symbols, where R is as large as needed, available for downloading. Then when a peer selects a piece to download from its neighbors, it can choose among the rarest R pieces instead of the rarest piece. In next section, we report the results of extensive simulations with the real world traces measured from the BitTorrent system to evaluate our RaptorQP2P protocol, which further demonstrates the effectiveness of our RaptorQ based protocol design.

V. Performance Evaluation

We conducted extensive simulations both to evaluate RaptorQP2P and to compare its performance fairly to BitTorrent. Our simulations of both RaptorQP2P and BitTorrent are driven by traces measured from the real BitTorrent system [17], 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 240 kB/s to 720 kB/s. The default number of

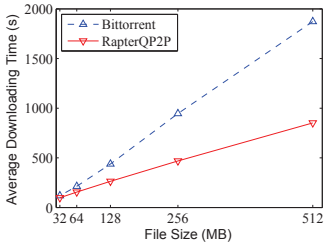

Fig. 4: Total downloading completion time as a function of file size for 1000 peers.

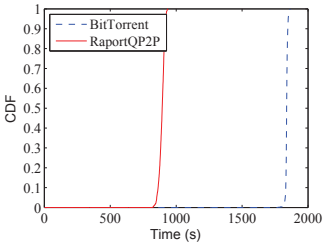

Fig. 5: CDF of individual peer's downloading completion time for a 512 MB file and 1000 peers.

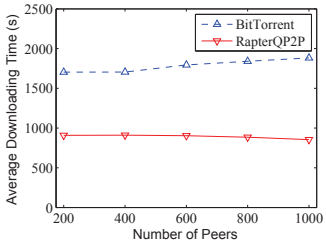

Fig. 6: Total downloading completion time as a function of the number of peers for a 512 MB file.

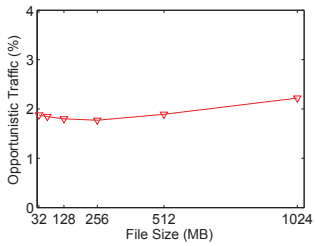

Fig. 7: The contribution of opportunistic transmissions to the whole system as a function of file size with 1000 peers.

peers is set to 1000 and the file size is 512 MB. Given that the block and symbol of RaptorQ are roughly equivalent to the piece and slice, respectively, of BitTorrent, to make a fair comparison, we adopt the typical setting of piece size (1600 kB) and slice size (16 kB) of BitTorrent and also use them for the block size and symbol size of RaptorQP2P. The number of neighbors in RaptorQP2P is set to the same as in BitTorrent and we also use tit-for-tat in RaptorQP2P for choking and unchoking peers. Other configurations are adopted from the typical settings used in [17][11][13][16]. We vary the number of peers from 200 to 1000 and the file size from 32 MB to 512 MB to further investigate the scalability of our solution. In the simulations, we focus on the download completion time, which is the amount of time required to download enough pieces (or blocks) to obtain the original file.

Fig. 4 shows the results of the total downloading completion time (i.e., the time for all the peers to get the original file) as a function of file size. As the file size becomes 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 512 MB, the time taken by RaptorQP2P is only 45.5% of the time taken by BitTorrent. This further demonstrates the effectiveness of our RaptorQ based protocol in distributing large files, which can be in great demand in 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 512 MB file, which is shown in Fig. 5. It is easy to see that all the peers using RaptorQP2P have much smaller downloading completion time than those using BitTorrent. In a real environment, peers may join the system at different times and their online/offline patterns can be totally different; this explains why the CDF curve of RaptorQP2P expands between 811 s and 932 s. However, no matter when a peer joins the system, all the peers using BitTorrent finish the downloading at times closer to each other (roughly around 1840 s). This is because under the network dynamics and peer churns, the well-known last piece problem of BitTorrent (i.e., it is often hard for a peer to identify the last missing piece for downloading) becomes more severe, as those peers with the last missing piece may dynamically leave the system and then rejoin later. On the other hand, due to the piece level encoding, the peers using RaptorQP2P have more flexible choices for their last missing pieces, since for an S-piece file, we now have roughly R (which is set equal to S in the simulation) options for the last missing piece.

We next examine how our solution scales with the number of peers, which is shown in Fig. 6. When the number of peers changes from 200 to 1000, the total downloading completion time of BitTorrent gradually increases with small fluctuations. This is because the peer traces are collected from the real system, where with more peers joining and leaving the system, more peer churns and flashcrowds may happen, not only degrading the overall performance but also bringing more fluctuations. RaptorQP2P, however, is very stable as the number of peers changes. This is because RaptorQ coding can automatically optimize symbol downloading among different neighbors even if they enter 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. We take a closer look at the ratio of data traffic delivered by our opportunistic transmission scheme, which is shown in Fig. 7. It can be seen that although there are small fluctuations when the file size is less than 256 MB, it shows an increasing trend as the file size grows large. To further verify this, we ran the simulation with a file size of 1024 MB, where the increasing trend becomes much clearer, achieving up to 2.22% for 1024 MB. This is because 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.

VI. Related Work

Since BitTorrent was first released in 2001, many researchers have been attracted to the peer-to-peer file distribution area. In [4], the author investigated the tit-for-tat and piece selection mechanisms, showing that the former can help prevent free riding and the latter can make the diversity of the

pieces in the whole system well balanced. In [1], the authors 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. In [13], the authors pointed out that the tit-for-tat can cause weak robustness.

Fountain Codes [10] are also called rateless erasure codes, which 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 are the first generation of fountain codes, where the symbol length can be arbitrary and the encoded symbols can be generated on-the-fly [9]. Recently, Raptor codes [14] were proposed as a type of more advanced fountain codes, which can conduct the encoding and decoding process in linear time. RaptorQ codes [7] 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 properties of the fountain codes, a number of proposals have been developed to use fountain codes to improve the performance [8][3][12], where the authors in [5] proposed to utilize Raptor Codes to accelerate P2P streaming. There have also been studies on utilizing fountain codes to accelerate P2P file sharing. In [15], the authors proposed to modify BitTorrent by using the LT codes. The modification, however, may result in duplicate encoded packets being received by a peer. Moreover, a coded packet forwarded by a peer may loop within the system, returning multiple times to the same peer. To address these issues and make data exchange more efficient, the authors in [11] 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, resulting in increased overhead for the data exchange, especially when the number of peers becomes large.

Different from these works, we propose a new protocol design using RaptorQ codes based on our modeling and analysis of the limitations of BitTorrent under various dynamic network environments, which also guides us to achieve better performance for P2P file distribution.

VII. Conclusion and Future Work

In this paper, we presented a novel RaptorQP2P protocol which applies the RaptorQ coding technique to P2P file distribution. Our design was motivated by our in-depth study on the limitations of BitTorrent under various dynamic network environments as well as lessons learned and documented in the literature. We conducted extensive and fair simulations of both RaptorQP2P and BitTorrent driven by the real traces measured from the BitTorrent system. The results show that our protocol can scale well with user population and achieve much better performance than BitTorrent, especially when dealing with large files. Indeed, our simulation results indicate that delivery of a 512 MB file to 1000 peers using RaptorQP2P required less than one-half the time required using BitTorrent.

We are currently conducting more simulations to further evaluate and improve our RaptorQP2P protocol. In addition, we have developed a Ruby-based interface to the RaptorQ API and installed the API and interface in a single-board computer-based testbed. We intend to use this testbed to implement and test variations on protocol designs before beginning tests in the Internet. Longer term plans include exploration of content delivery in wireless networks and other potential applications of RaptorQ technology, such as datacenter networks.

Acknowledgment

This research is partly supported by a Start-up Grant from the University of Mississippi, a NASA EPSCoR Program under grant NNX13AB31A and grant NNX14AN38A, as well as a Chancellor's Small Grant and a Start-up Grant from the University of Minnesota at Duluth. The authors thank Michael Luby of Qualcomm Technologies Inc. for his general descriptions of the properties of the RaptorQ codes and his helpful comments on this paper.

References

[1] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Networks Performance Mechanisms," in *INFOCOM*, 2006.
[2] BitTorrent, http://www.bittorrent.com.
[3] C. Bouras, N. Kanakis, V. Kokkinos, , and A. Papazois, "Embracing RaptorQ FEC in 3GPP Multicast Service," *Wireless Networks*, vol. 19, no. 5, pp. 1023–1035, 2013.
[4] B. Cohen, "Incentives Build Robustness in BitTorrent," in *IPTPS*, 2003.
[5] P. M. Eittenberger, T. Mladenov, and U. R. Krieger, "Raptor Codes for P2P Streaming," in *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2012.
[6] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest First and Choke Algorithms Are Enough," in *ACM IMC*, 2006.
[7] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery," in *IETF*, 2011, IETF RFC6330.
[8] M. Luby, "Best practices for mobile broadcast delivery and playback of multimedia content," in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, 2012.
[9] ——, "LT Codes," in *IEEE Symposium on Foundations of Computer Science*, 2002.
[10] J. MacKay, "Fountain Codes," *IEE proceedings-Communications*, vol. 152, no. 6, pp. 1062–1068, 2005.
[11] A. Magnetto, S. Spoto, R. Gaeta, M. Grangetto, and M. Sereno, "Fountains vs Torrents: the P2P ToroVerde Protocol," in *IEEE/ACM Annual International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2010.
[12] B. Miguel, C. Toh, C. T. Calafate, J.-C. Cano, and P. Manzoni, "RCDP: Raptor-based content delivery protocol for unicast communication in wireless networks for ITS," *Wireless Networks*, vol. 15, no. 2, pp. 198–206, 2013.
[13] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in BitTorrent?" in *USENIX NSDI*, 2007.
[14] A. Shokrollahi and M. Luby, "Raptor Codes," *Foundations and Trends® in Communications and Information Theory*, vol. 6, no. 3-4, pp. 213–322, 2011.
[15] S. Spoto, R. Gaeta, M. Grangetto, and M. Sereno., "BitTorrent and Fountain Codes: Friends or Foes ?" in *International Workshop on Hot Topics in Peer-to-Peer Systems (HotP2P)*, 2010.
[16] H. Wang, F. Wang, and J. Liu, "On Long-Term Social Relationships in Peer-to-Peer Systems," in *IEEE/ACM IWQoS*, 2011.
[17] ——, "Accelerating Peer-to-Peer File Sharing with Social Relations: Potentials and Challenges," in *INFOCOM*, 2012.