中央研究院
資訊科學研究所
Institute of Information Science, Academia Sinica • Taipei, Taiwan, ROC

TR-IIS-06-015

Bee: A Best Effort Peer-to-Peer Delivery Protocol for Internet Data Dissemination Application

Chi-Jen Wu, Cheng-Ying Li, Kai-Hsiang Yang and Jan-Ming Ho

November 30, 2006 || Technical Report No. TR-IIS-06-015

http://www.iis.sinica.edu.tw/LIB/TechReport/tr2006/tr06.html

Bee: A Best Effort Peer-to-Peer Delivery Protocol for Internet Data Dissemination Application

Chi-Jen Wu, Cheng-Ying Li, Kai-Hsiang Yang and Jan-Ming Ho, *Member, IEEE*

***Abstract*— How to rapidly disseminate a large-sized file to many recipients is a fundamental problem in many applications, such as updating software patches and distributing multimedia content. In this paper, we present the Bee protocol, which is a best-effort peer-to-peer file delivery protocol aiming at minimizing the maximum dissemination time for all peers to obtain the complete file. Bee is a decentralized protocol that organizes peers into a randomized mesh-based overlay and each peer only works with local knowledge. We introduce a slowest peer first strategy to boost the speed of dissemination, and design a topology adaptation algorithm that adapts the number of connections based on upload bandwidth capacity of a peer. Moreover, Bee is designed to support network heterogeneity and deal with the flash crowd arrival pattern without sacrificing the dissemination speed. We present a lower bound analysis and experimental results on the performance of Bee in terms of dissemination time and show that its performance can approaches the lower bound of the maximum dissemination time.**

***Index Terms*— Peer-to-peer System, Content Delivery, Data Dissemination.**

I. Introduction

HOW to rapidly, reliably and efficiently distribute a large file across a wide-area network has become an interesting problem in the peer-to-peer (P2P) research community and some real applications, such as updating the software patches of Massively Multiplayer Online Games (MMOG) [1] and operation systems [2] in a flash crowd arrival pattern. Suppose that a large file is initially held at a single server and we have to disseminate it to other N peers, how can we minimize the time it takes for all peers to have the complete file?

In the Internet environment, several characteristics make it hard to design a scalable protocol that configure resources (such as computing power and network bandwidth) for disseminating data to a large number of clients, including: **1) Scalability:** The number of participating nodes must be in the thousands or even more. **2) Unstable:** The behavior of participating users is characterized by the dynamics with which the nodes join, leave, and rejoin the system at arbitrary time, making it difficult, if not impossible, to maintain an infrastructure among the large number of participating nodes. **3) Heterogeneity:** The resources, such as bandwidth, computing power of participating nodes often are heterogeneous, which make it difficult to make a schedule in polynomial time. **4) Network dynamic:** Routers, links in the Internet environment and the nodes may fail, incurring more transmission cost and longer transmission time to deliver the large amount of contents. For these reasons, it is difficult to maintain a large-scale data dissemination system that requires a large amount of communications among the requesting nodes.

Native IP multicast is an efficient and scalable solution for the data dissemination problem [3]. With native IP multicast technique, an application can deliver content to a set of receivers efficiently. This technique reduces the network traffic by simultaneously delivering a single data stream to thousands of end nodes. However, a number of considerations, including scale, reliability and business policy of ISPs have limited the widespread success and availability of IP multicast across the Internet infrastructure.

Since the native IP multicast solution has not generally received widespread deployment, a number of efforts focus on building P2P content delivery architectures [4-8] to address the data dissemination problem. The main idea is that a source can transmit data to a set of peers by TCP connections, and these peers across the Internet act as intermediate routers to deliver data over a predefined network, such as a mesh or multiple trees at the application level. Moreover, in these P2P architectures, data are usually divided into M parts of equal size, each being called a block. A peer may download any one of these blocks either from the server or from a peer who has downloaded that block. Besides, many researchers, including [9-10], have studied the performance of these P2P architectures, and show that the P2P architecture is both efficient and scalable, even it lacks a centralized coordination and scheduling mechanism.

A famous P2P content delivery architecture is the BitTorrent system [4], which is one of the pioneers of the file dissemination systems, and has become a prominent Internet application, both in terms of user popularity and traffic volumes [11]. However, BitTorrent is designed to minimize the dissemination time of each peer egoistically. On the other hand, other recent studies (including Slurpie[5], Bullet [6], SplitStream [7] and Crew [8]) have proposed to construct and maintain an overlay network of multiple trees or a random mesh to deliver data from a single server.

In this paper, we are interested in a specific version of the data dissemination problem that arises in current Internet applications, such as updating software patches or distributing multimedia contents, especially for the flash crowd arrival pattern. In the rest of the paper, we then addresses the following two questions:

i) what is the lower bound of the data dissemination problem?

Manuscript received November 15, 2006. This work was supported by the the Institute of Information Science, Academia Sinica.

Chi-Jen Wu, Cheng-Ying Li, Kai-Hsiang Yang and Jan-Ming Ho are with the Institute of Information Science, Academia Sinica, Taiwan. (e-mail: {cjwu, cyli, khyang, hoho}@iis.sinica.edu.tw)

ii) is it possible to design a decentralized protocol that can achieve or approach to the lower bound?

We begin by giving a formal definition of the data dissemination problem. We then derive the lower bound of the dissemination time for the problem. Note that this lower bound applies to both homogeneous and heterogeneous networks.

In addition, we present a decentralized protocol, called "Bee", to address the data dissemination problem. Bee is designed from a *best-effort* service concept, to increase both the system throughput and peer concurrency. In the *best-effort* service concept, a peer allocates bandwidth for all its neighbors and attempts to serve all of them. Basically, Bee is very similar to the BitTorrent system. Peers in Bee begin by self-organizing into a random mesh, and download blocks from as many neighboring peers as possible. In the Bee protocol, we present a slowest peer first strategy and a topology adaptation algorithm to maximize the speed of dissemination. Under the slowest peer first strategy, a peer always transmits blocks to the neighbors that have fewest number of downloaded blocks. Bee protocol also embeds a topology adaptation algorithm for a peer to adapt number of connections to its neighbors based on its own upload bandwidth. Moreover, our experimental results show that the dissemination time of the Bee protocol approaches the lower bound of the data dissemination problem that is derived in this paper for both homogenous and heterogeneous network environments. To the best of our knowledge, this work is the first that systematically studies the effects of P2P architecture with respect to minimizing data dissemination time in data networks.

The rest of the paper is organized as follows. In Section II, we first define the data dissemination problem. Section III describes an overview and design details of the Bee protocol. We give a detailed analysis of Bee protocol in section IV. Section V explains our simulation methodology and presents the performance results of the simulation study. In section VI, we discuss related work. Section VII concludes this paper with a summery of the main research results of this study.

II. Data Dissemination Problem

In this section, we formally define the data dissemination problem, and provide a lower bound of the problem.

Let us consider the problem of disseminating a file F to a set of n peers, $\mathcal{N} = \{1, 2, \ldots, n\}$. We also assume that a peer leaves the system once completely receiving the file. Let S be the server (we called it "seed" in the rest of this paper) that has the file F in the beginning, and let $Size(F)$ denote the size of file F in bytes. Each peer $i \in \mathcal{N}$ in this system has its upload capacity U_i and download capacity D_i. Both U_i and D_i respectively represent the upper bound of the upload and download bandwidth utilization of peer i. We also assume that $U_i \leq D_i$, to model the state-of-the-art Internet technologies, such as ADSL or Cable modems. Due to the asymmetric nature of these network technologies, D_i is usually 3 to 5 times higher than U_i in practice. Let $t_i(F)$ denote the time it takes for peer i to receive the complete file F. Note that $t_i(F)$ denotes the time interval starting at the time peer i sends its request to the server and ending at the time it receives the entire file F. Before formally defining the problem, we define the following two performance metrics first.

Definition 1 (Average Dissemination Time, ADT(F)):

$$ADT(F) = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} t_i(F).$$

Definition 2 (Maximum Dissemination Time, MDT(F)):

$$MDT(F) = \max\{t_i(F)\}, i \in \mathcal{N}.$$

Assume that the server and all n peers exist in the system from time $t = 0$, then $MDT(F)$ is the time it takes for all peers to finish receiving the complete file F. Now we define the data dissemination problem as follows.

Definition 3 (File Dissemination Problem): Given a server and n peers in the system, and each peer i has the upload capacity U_i and download capacity D_i, where $i = \{1, 2, \ldots, n\}$, the problem is to find a transmission scheme $\mathcal{M}$ to minimize the $MDT(F)$. According the definition 2, the problem can be treated as a min-max problem as follows.

$$\min\{MDT(F)\}. \tag{1}$$

Note that the data dissemination problem is somewhat different from the broadcast network problem where a node wants to broadcast a message to all other nodes as fast as possible. Unlike the data dissemination problem in which a peer is allowed to transmit and receive data simultaneously subject to the upload and download bandwidth constraints, in the broadcast network problem, a node can either transmit messages to or receive messages from another node, but not both. Many researchers have studied the broadcast network problem in homogenous networks for more than 20 years, and its optimal solution was presented in 1980 [12]. However, when nodes have heterogeneous bandwidths, the problem becomes $\mathcal{NP}$-hard [13]. So unless $\mathcal{P} = \mathcal{NP}$ there is no polynomial time algorithm to find an optimal solution. In year 2006, Deshpande et al. [14] proposed two centralized heuristics for the broadcast network problem in a heterogenous network environment.

A. Ideal Dissemination Time (Lower Bound)

In this section, we focus on studying the lower bound of the dissemination time, which is also denoted as "ideal dissemination time" in this paper, in a heterogenous network environment. We assume that peers are highly cooperative that is, each peer is willing to forward data to other peers as fast as possible. Let us denote the actual amount of data uploaded by peer i as f_i, where $f_i \leq U_i \times t_i(F)$ and those peer i receives in return as r_i, where $r_i \leq D_i \times t_i(F)$. Without loss of generality, we may assume that, the total amount of download data must be equal to the total amount of upload data for the seed and all peers. Hence, we have the following equation.

$$f_s + \sum_{i=1}^{n} f_i = \sum_{i=1}^{n} r_i. \tag{2}$$

Since we are interested in estimating the lower bound of the dissemination time, we assume that upload capacity of each peer i is assumed to be fully utilized, i.e., we have $f_i = U_i \times t_i(F)$. Besides, the total amount of download bandwidth must be equal to $n \times Size(F)$, because all peers have the entire file F at the end. Then, we can extend the Eq. (2) as follows to deal with the ideal dissemination time for the general case.

$$U_s \times t_s(F) + \sum_{i=1}^{n} U_i \times t_i(F) = n \times Size(F). \quad (3)$$

Here, we have a min-max problem with its objective function in Eq. (1) subject to the constraints given by Eq. (3). Since the constraint is a linear equation, or more specifically, a hyperplane in $(n+1)$-dimension, it is not difficult to show that the optimal solution for the constrained optimization problem can be obtained if and only if when all t_i are the same, i.e.,

$$t_s(F) = t_1(F) = t_2(F) = \cdots = t_n(F). \quad (4)$$

We leave details of the proof to interested readers. Applying this results to Eq. (3), we then have a lower bound of $MDT(F)$, denoted by $\overline{T}(F)$, for file F, as follows.

$$\overline{T}(F) = \frac{n \times Size(F)}{U_s + \sum_{i=1}^{n} U_i}.$$

Now, we are ready to present the following lemma.

Lemma 1: Let $T^*(F)$ denote the $MDT(F)$, of a feasible schedule of the data dissemination problem for a given file F. Then we have:

$$T^*(F) \geq \max\left\{\overline{T}(F), \frac{size(F)}{U_s}, \frac{size(F)}{\min\{D_i\}}\right\}, \quad (5)$$

where the right-hand right of Eq. (5) is the lower bound of the dissemination time in any algorithm for the data dissemination problem.

Proof: Note that the lemma merely says that $T^*(F)$ must be greater than 1) the lower bound we derived in the above; 2) the time for the seed to transmit the file F; 3) the time for the slowest peer to download the file F. ■

This analysis indicates that if someone wants to design a data dissemination protocol to approach the lower bound in a general network environment, two prerequisite keys should be considered: 1) the protocol should enforce the peers to leave the system at almost the same time, and 2) the protocol should always fully utilize the upload bandwidth of each peer. Base on these observations, we present our design of the Bee protocol in the following section.

III. BEE DESIGN

In this section, we present the Bee protocol to approach the ideal dissemination time that we derived in Section II. In the Bee protocol, like other P2P content delivery systems, the content is divided into many fix-sized blocks $B_i, i = \{1, 2, \ldots, m\}$, that is the smallest transfer unit[1] in the system. In addition, each block should be small enough to provide a fine granularity of striping, and it should also be sufficiently large as to fully utilize the network resource until complete termination of the file transfer. We chose 256KB for the block size, which is the same as that used in other P2P content delivery protocols. A successful completion of the transmission consists of receiving all blocks. In the following, we start with an overview of the Bee protocol, followed by the detailed descriptions of its various components.

[1]We do not ponder on the round off issues that may make the last block smaller.

A. Overview of Bee

At a high-level concept of overview, Bee constructs a random mesh overlay among a set of participating peers. Fig. 1 illustrates the scheme of a Bee system. Suppose that a large content is announced from a single seed, and particularly we assume that $U_s \geq U_i$ in the system, and a lot of peers want to download the content (file) at the same time. Each peer gets into contact with a centralized well-know register server and retrieves a *contact list* of an uniform random subsets of all peers in the system. The size of contact list is a small constant, say 40. The register server is a powerful and stable server which keeps the track of IP addresses of all peers. Seriously, the register server may suffer the scalability problem, especially when the number of participating peers is very large. However, there are some techniques of distribution systems, e.g. cluster systems, can address this problem, we do not discuss the problem here.

Based one the contact list returned from the register server, the peer discovers other peers in the system, and exchanges update messages with them. The update message contains a *bit string* about which blocks are available, and the bit string can be used to coordinate the block requesting decisions without global information. After exchanging update messages, a peer could send requesting block messages to all peers in the contact list, and downloads blocks while uploading blocks it owns to other peers simultaneously. In this way, the load is distributed among all peers in the system. A peer in Bee system will periodically ask the register server for a new contact list to maintain the random mesh overlay, or when it has no block to download from its connected peers.

The key components in the Bee system include (1) a slowest peer first strategy for finding suitable peers to upload blocks,

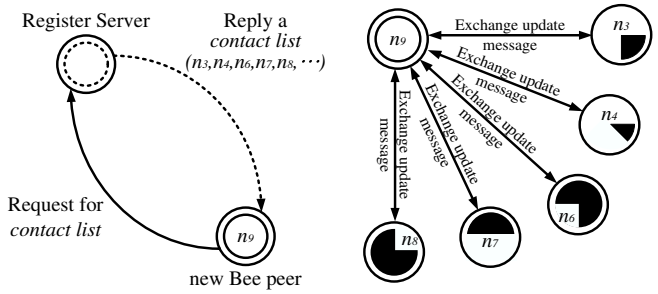


Fig. 1. A scheme of a Bee system: The peers in Bee form a random mesh, discovery the mesh neighbors and exchange the update messages.

(2) a block selection strategy for requesting blocks from neighbor peers, and (3) a topology adaptation algorithm for adapting the number of connections according to a peer's network capacity. The detailed descriptions of these components are presented as follows.

B. Slowest Peer First Strategy

In this section, we describe the slowest peer first strategy and the procedure that each peer performs. Overall, a good peer selection approach would be one which neither requests peers to maintain global knowledge, nor communicate with a large number of peers, but the one which is able to find peers having blocks the peer needs. Hence, our focus is the development of a decentralized mechanism, within which each peer learns its nearby peers' statuses (with local knowledge) and selects a suitable peer to upload blocks in order to minimize the maximum dissemination time of the system.

The design principle of the slowest peer first strategy is to keep all the upload capacity of peers full of data. This means that a peer always can find some peers to upload blocks to exhaust its upload capacity. Based on the slowest peer first strategy, a peer i always picks a slowest downloading peer among the contact list of the peer i, where the slowest downloading peer is the peer that has the least number of blocks. Consequently, the peer i always can upload blocks to the picked peer, because there is a high probability that the peer i has some blocks that the picked peer does not have. In this point of view, this peer selection strategy makes our Bee protocol to be able to utilize the upload bandwidth of each peer as much as possible.

The operation of the slowest peer first strategy is very simple but efficient. We use an illustration of the slowest peer first strategy in Fig. 2 to present this simple idea. After a peer joins into a Bee system, it periodically sends the requesting block messages to the peers in the contact list for downloading the blocks it lacks. When a peer starts to upload blocks to other peers, it maintains a *working set*, and the peer tries to use its network capacity as much as possible to upload blocks to the peers in its working set. The working set is a set of peers selecting from the contact list over a period of time. The size of working set is adapted by a topology adaptation algorithm that we will discuss later. We show the pseudo code of the slowest peer first algorithm in Algorithm 1.

Algorithm 1 Pseudo Code of the Slowest Peer First Strategy

```
1: BEGIN
2: WorkingSet[] ← Null
3: for j ← 0 to Sizeof(WorkingSet[]) do
4:    Pick the slowest peer p ∈ Contact List
5:    WorkingSet[j] ← peer p
6:    j ← j + 1
7: end for
8: return WorkingSet[]
9: END
```

The advantage of the slowest peer first strategy is to enforce peers to download blocks at approximately the same speed,

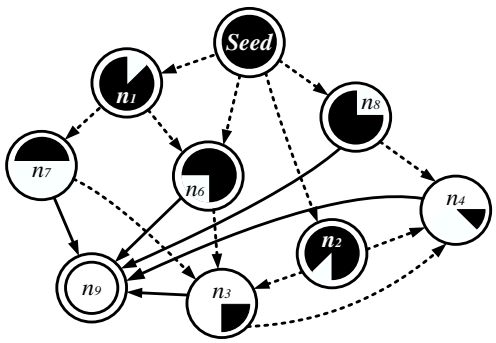


Fig. 2. An illustration of the slowest peer first strategy: The shaded content within a peer represents the percentage of the file that a peer has downloaded. Using the slowest peer first strategy, a peer always forwards blocks to the slowest peer among its contact list.

and this behavior can significantly diminish the MDT that we defined in Section II. However, the disadvantage of the strategy is that the faster downloading peers will be delayed for the slower downloading peers. Let us think if some faster downloading peers leave the system early, the MDT of the system will be prolonged undoubtedly, recalling the inferences of Eq. (3) and (4) in Section II. Hence, this problem can be addressed by the proposed slowest peer first strategy fundamentally.

C. Block Selection Strategy

Once a peer establishes connections with its neighboring peers, it needs to determine which blocks to request from which peers, based on the local knowledge (the available blocks in all peers among the contact list). The Bee protocol employs the local rarest first strategy for choosing new blocks to download from neighboring peers. The local rarest first strategy is proposed in BitTorrent protocol, and it can prevent the last block problem and increase the file availability in a BitTorrent system. The main advantage of the local rarest first strategy is to overcome the last block problem by favoring rare blocks. This strategy equalizes the file block distribution to minimize the risk that some rare blocks are lost when peers owning them fail or depart the system. Bharambe *et al.* [16] study the local rarest first strategy by simulations and show that this strategy can address the last block problem efficiently. Another advantage of the local rarest first strategy is to increase the probability that a peer is useful to its neighboring peers because it owns the blocks that others do not have, and thus it helps diversify the range of blocks in the system. For above reasons, the Bee protocol also uses the local rarest first strategy to select the requesting blocks. Moreover, applying this strategy, Bee can reduce the complexity of protocol communication, because each peer only needs to maintain local information of each block.

Next, we represent the computing model of the local rarest block strategy. Suppose that B denotes the set of overall blocks in the file being distributed, and G_i and M_i are the set of blocks that peer i has already gotten and is still missing, respectively (where $B = M_i \cup G_i$ and $M_i \cap G_i = null$).

Similarly, $M_i \cap M_j \neq null$ means that the peers i and j both are missing at least a block m, and they try to find and download the block m. The requesting peer r selects the rarest block $\hat{m} \in (G_j \cap M_r)$ among those that it misses and one of its neighbors, say peer j, held. The rarest block is computed from the number of each block that held by the neighbors of requesting peer r. More precisely, we use the following computing function for the local rarest block strategy:

$$\forall\ m \in (G_j \cap M_r)$$

$$\mathcal{LRF}(\hat{m}) = \max \left\{ \sum_{i \in \textit{Contact List}} |m \cap M_i| \right\}.$$

The above function $\mathcal{LRF}(\hat{m})$ represents a computing model of the local rarest block strategy for a peer r. For each block $m \in (G_j \cap M_r)$, a peer r calculates the number of neighboring peers missing the block, and it chooses the block with the maximum value for requesting download.

D. Topology Adaptation Algorithm

The design of Bee explicitly takes into account the capacity heterogeneity associated with each peer in the P2P network. Bee is designed to adapt to different network environments by a topology adaptation algorithm. In general, the available bandwidth estimation is a non-trivial problem in practical network applications, so it is hard to decide how many upload connections a peer should have in the Bee protocol. However, using a fixed number of upload connections will not perform well under a wide variety of peers' network capacities. Hence, the Bee protocol employs a simple flow control algorithm that attempts to dynamically change the maximum number of upload connections (indicated the size of working set in Bee) according to the upload capacity of each peer.

For the sake of simplicity, we do not use the network bandwidth estimation techniques [20-21] to determine the precise upload capacity of each peers. Instead, we assume that the user can input a coarse grained bandwidth estimate, such as the form ADSL, Cable, T3, etc, that provides an initial maximum upload capacity estimate U_i. In addition, we assume that peers (including the seed) have limited upload/download bandwidth but the Internet backbone is assumed to have infinite bandwidth, as illustrated in Fig. 3. This assumption is reasonable because the previous study [22] showed that the Internet backbone indeed has very low utilization and the bottleneck almost happens at the parts near the end hosts. Based on the two assumptions, we can develop a simple topology adaptation algorithm for the Bee protocol.

A simple adaptation approach is to set the upload rate for each upload connection to the same value, say rate r, for all peers. Hence, if a peer i has maximum upload capacity of U_i, it establishes $k = \lceil \frac{U_i}{r} \rceil$ connections, where $r \leq U_i$, $\forall\ i \in \mathcal{N}$. So in the Bee protocol, each peer establishes k concurrent upload connections among its working set, and intuitively a peer can quickly upload the blocks it holds to other peers.

The basic idea behind this approach is that by serving k different peers simultaneously, the peer can fully utilize its upload capacity and thus maximize its contribution to the system throughput. For example, a peer with higher capacity might establish ten or more connections than a peer with lower capacity. However, the intuition is not always correct. When a smaller upload rate r is chosen, the lower bandwidth dedicates to each connection. As a result, a smaller value of r might slow down the distribution rate for blocks. We will consider different values of the upload rate r in our analysis.

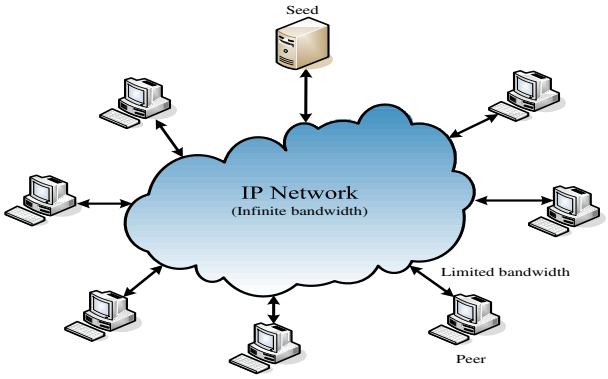


Fig. 3. The simplified network-capacity framework

IV. Protocol Analysis

In this section, we will describe informally that the dissemination time in the Bee protocol can achieve the ideal dissemination time with a very high probability, and we also provide a complexity analysis for the Bee protocol in terms of the scalability, efficiency, communication complexity and the fairness issues. During the analysis, we assume that all $\mathcal{N}$ peers join the system at the same time and all the communications between peers are reliable. We also assume that peers do not leave the system either voluntarily or due to failures.

A. Scalability

The proposed Bee protocol in section III is very scalable. A peer only needs to maintain a random overlay mesh with a constant number of connections, regardless of the size of the system (the total number of peers in the system). This implies that each peer only connects to a small number of other peers, so the load in each peer is very slight.

However, one possible concern is the scalability of the register server in Bee. The resigner server in Bee serves as the same role as the tracker in BitTorrent or the rendezvous point in some application overlay multicast systems [23]. In practice, the resigner server stores the IP addresses of all peers in the system; assume that the number of peers is one million, the amount of storage is approximate to 4 megabytes (4 bytes for an IP address multiply by the number of peers), so any usual machine can store the amount of storage for one million peers. In addition, the functionality of the register server can be distributed by the cluster system techniques, so that service load can be shared to many servers. We believe that the register server will not be a critical problem to limit the scalability of our Bee protocol.

B. Efficiency

In this section, we briefly discuss that the dissemination time in the Bee protocol can achieve the theoretical lower bound with a high probability. We do not provide a theoretical proof to show the optimality of the Bee protocol due to the inherent difficulty of any heuristic-driven distributed protocols, such as BitTorrent. To the best of our knowledge, we are not aware of any theoretical results to prove that a decentralized protocol can achieve the lower bound. In [24], Wu *et al*. gave a centralized scheduling algorithm to minimize the dissemination time in an overlay with all knowledge of each peer's network capacity. However, their centralized solution only works in a static network environment, and they also do not consider the dynamic behavior of peers, such as joining and leaving, which are common in real P2P systems.

We now concisely analyze the Bee protocol. First, we assume that the goal of the Bee protocol is to disseminate a file F from a seed to a number of receivers under the constraints of $\frac{size(F)}{U_s} \leq \overline{T}(F)$ and $\frac{size(F)}{\min\{D_i\}} \leq \overline{T}(F)$. When the two constraints hold, neither the seed nor the slowest peer does not become the bottleneck in the system. In addition, we also assume that the blocks are uniformly distributed among peers, which is caused by the local rarest first strategy and no any communication overhead and delay in the Bee protocol.

Recall the analysis in the section II, we introduced two important design principles for a data dissemination protocol to achieve the theoretical lower bound. The first one is that all peers should leave the system at the same time as much as possible, and each peer has to utilize its upload bandwidth as much as possible. For the first design principle, the slowest peer strategy in Bee forces peers to progress at approximately the same download speed. Second, all peers in Bee always try to fully utilize the upload bandwidth by the topology adaptation algorithm, and thus each peer maximizes its contribution to the system throughput. However, at the beginning of the system, only the seed has the file, so it is impossible to utilize the upload bandwidth of each peer. We call the period of time for the system so that each peer has enough blocks to exchange as the start-up time of the system. Assume that a block size is $256KB$, so the start-up time approximately equates to $\frac{256KB}{r} \times \log n$, where r is the upload rate and n is the number of peers in the system. This start-up time can be ignored when the distribution file is very huge. Moreover, in the system, a peer only sends out a block when it already received a request from a peer, so no blocks are forwarded without requesting, and a peer will not receive duplicated blocks.

Here, we provide a simple example to understand the operation in Bee. In the homogenous network environment, the upload capacity k of each peer is equivalent, assume $k = 5$ ($\frac{U_i}{r} = 5$). If the number of peers is n, the number of incoming connections per peer is in average 5 ($= \frac{n \times k}{n}$), because the overlay mesh in the Bee system is randomly constructed. Based on the same concept for the heterogeneous network environments, each peer in Bee would have the same number of incoming connections in average. Hence, it is easy to understand that the Bee system can force peers to progress at approximately the same download speed and achieve the lower bound both in the homogenous and heterogeneous network environments.

C. Communication Complexity

We now discuss why the control overhead in the Bee system is very slight. The communication complexity is defined as the number of control messages required for a peer to maintain an overlay topology and to request blocks for downloading. Here, let us consider the initial state of the system, where many peers join in the system and connect to each other to establish a mesh, and then all peers start to download blocks from one another. After that, other peers may join and leave and the overlay mesh may likely change, but the communication complexity of those peers is almost the same as that of the peers in the initial state.

When a peer joins into the Bee system, it first contacts the register server for requesting a contact list, and during the dissemination process, it may contact with the register server several times for updating the contact list. Assume that a peer requesting a contact list once needs the constant overhead, say $\mathcal{C}$, so the overhead for requesting contact lists is $O(\mathcal{C})$. After obtaining a contact list, a peer first needs to exchange its state with all its neighbors, and does this operation during the dissemination process several times. Assume that the communication overhead to exchange states with all neighbors once is a constant value $\mathcal{D}$, so the overall overhead for the state maintenance can be estimated by $O(\mathcal{D})$. After that, a peer starts to send out requesting messages to its neighbors, and the total number of requesting messages is equal to the number of blocks, m, because a peer only sends out a message for a block.

According to the analysis, total communication complexity for a peer is O($\mathcal{C} + \mathcal{D} + m$), thus the communication overhead for a peer in the Bee protocol is relatively slight.

D. Fast Dissemination vs. Fairness

In general, to make the dissemination process as fast as possible, peers need to contribute (via uploading) bandwidth as best as it can. This implies that a fast dissemination protocol needs to keep high capacity peers to stay in the system to serve others, thus fairness is not a major concern. In other words, it is clear that there exist scenarios where there are tradeoffs between fast dissemination and fairness. Some protocols, such as BitTorrent, FOX [25], have some mechanisms of fairness to make all peers both give and take equitably. BitTorrent encourages fairness by using the tit-for-tat (TFT) as the peer selection strategy, so that peers can share blocks with one another alike. FOX provides a theoretically optimal download time when all peers are selfish, but it is a centralized protocol and only works in homogeneous networks.

On the contrary, our Bee system is designed to minimize the dissemination time, the fairness issue for the Bee protocol is not the first concern. In particular, we believe that there is no "one fits all" protocol, as each of them offers various tradeoffs and may prove most adequate for specific deployment scenarios.

V. Performance Evaluation

To understand the performance of our protocol, we built a discrete-event simulator to simulate the distribution of a large file from a server to a large number of peers in the Bee protocol, so that we can comprehensively study its performance in a wide range of network configurations. For example, we may simulate a large-scale network environment with heterogeneous link capacity. Our simulator is based on the paper [16] which is the first to study the performance of the BitTorrent-like systems by simulations. In our simulator, the network model associates a download link and an upload link bandwidth with each peer in order to make it suitable for modeling asymmetric access networks.

However, the computational complexity of even this simple model is still complicated. Hence, we decide to simplify the network model in a higher level of abstraction way that can significantly improve the scalability of our simulation. We simplified our network model in the following ways: first, we do not consider any shared bottleneck link in the core network, so we assume the core network has infinite capacity. This assumption is reasonable because a previous study [22] showed that the Internet backbone indeed has very low utilization and the bottleneck links almost happen at the links near to end hosts. In addition, we also do not consider network propagation delay and the dynamics of TCP connections. Instead, we assume that all TCP connections passing a specific link share the link bandwidth equally, so the dissemination time of a protocol is dominated by the time to transfer data. Note that if the download capacity of receiver is the bottleneck, the spare available upload capacity of sender will be used by its other uploading connections. Finally, the endgame model [4] of BitTorrent is not simulated because it only works for a small percentage of the download time when the download time is very lengthy. This simplification of BitTorrent adversely may have a tiny impact on the dissemination time, but not a decisive one.

Obviously, there are some important network parameters that we do not simulate, such as locality properties in constructing the overlay, cross traffic impact, or malicious users. Although the above simplifications may have impact on our experiment results, we wish our simulation is able to capture some of the most important properties for the design philosophy essentially.

A. Road-map of Simulations

We made our simulations to compare the dissemination time in the Bee system with the lower bound of dissemination time and the required time in the BitTorrent system. We implemented the BitTorrent system according to the detail overview of BitTorrent [4, 16]. Besides, we consider three network scenarios to evaluate our protocol, each representing a different degree of heterogeneity in their upload/download capacity and the peer join pattern is set to flash crowd. Note that the lower bound of dissemination time in each scenario is calculated by Eq. (5) in section II. In each network scenario, we explore the impact of network size, upload bandwidth of seed and the parameter r in our Bee protocol.

TABLE I
The upload/download bandwidth distribution

Network Type	Downloadlink	Uplink	Fraction
Homogeneous	1500kbps	384kbps	100%
Heterogeneous	1500kbps	384kbps	50%
	3000kbps	1000kbps	50%
More heterogeneous	784kbps	128kbps	20%
	1500kbps	384kbps	40%
	3000kbps	1000kbps	25%
	10000kbps	5000kbps	15%

The bandwidth distribution of each network condition is presented in Table I. First, we consider a homogeneous setting where all peers have the same upload/download capacity. Then the heterogeneous network has two types of peers, one of which has a higher upload/download capacity than the other. And then, a more heterogeneous condition with four types of peers is considered, and this network setting is the actual peer bandwidth distribution which is reported for Gnutella clients [32].

Moreover, we study the join patterns besides the flash crowd scenario. In particular we consider cases where peers with various join ratios to evaluate the impacts of join rates for the Bee and BitTorrent systems in the more heterogeneous network conditions. We also present the results of a realistic join pattern that derived from a tracker log for Redhat 9 distribution torrent of a BitTorrent system [15].

Unless otherwise specified, we use the following settings in our experiments. First, we used a file size of 200MB with a block size 256KB (so a file contains 800 blocks). The seed's upload capacity is 6000Kbps. The number of contact list is 40 in both the Bee and BitTorrent system, and the maximal number of concurrent upload connections per peer is 5 in BitTorrent setting (including the optimistically unchoked connection). Then the number of initial seeds is only one in all of our experiments.

B. Homogeneous Environment

We start by comparing the performance of Bee with that of BitTorrent protocol in the homogeneous environment. We use the default settings as mentioned in above section. All peers join the system at the initial stage, and leave the system when they finish their downloading. First, we study the influence of the parameter r in our Bee protocol, and then evaluate the impact of network size, and upload bandwidth of seed. In this scenario, the lower bound is 4035 according to Eq. (5). Moreover, we do not show the upload link utilization, due to the upload link utilization of this two protocols is over 90%, which means the overall upload capacity of network is almost fully utilized.

Fig. 4 depicts the dissemination time (both ADT and MDT) with 95% confidence intervals from the simulations as we vary the parameter r of the Bee protocol in a network with 1000 peers. In Fig. 4, x axis shows different uploading rate r,

and y axis shows the ADT and MDT. It is easy to see that when r grows, the MDT increases in a logarithmic manner and the ADT of various r is a smooth variation. These results show that the parameter r has a little effect on the performance of Bee, and the dissemination time of Bee is close to the lower bound. Based on this experiment, we choose the uploading rate $r = 25$ in the following experiments.

Next, we consider the impact of seed bandwidth on the performance of Bee and BitTorrent. We fix the number of peers that join the system to 1000 in the initial stage. Here, we use a normalized MDT metric which is the MDT dividing the lower bound, and a normalized ADT is also computed by the same way. Fig. 5 shows the normalized MDT in the Bee and BitTorrent when the bandwidth of a seed varies from 1000 Kbps to 6000 Kbps. Here, we only represent the MDT due to the ADT is very similar. In Fig 5, We can easily to see that the dissemination times in Bee and BitTorrent are close to the lower bound. However, when the seed bandwidth downs to 1000 Kbps, the performance of our Bee system decreases largely. The reason for this phenomenon is that the seed becomes the bottleneck. This implies that the efficacy of Bee is limited by the seed bandwidth extraordinarily, and BitTorrent is a very tolerant protocol for poor seed capacity network scenarios.

Another important measure is to study the scalability of Bee by increasing the network size. We use different number of peers from 500 to 5000 in experiments, and all peers join system at the initial stage and remain in the system until they have a complete file. Fig. 6 (a) and (b) show the normalized ADT and the normalized MDT, respectively. The difference of ADT between Bee and BitTorrent is not vary much, but the MDT between Bee and BitTorrent exists a gap regardless of network size. Moreover, the difference ratio between Bee and the lower bound is only 1.1 at a network with 5000 peers. Fig. 6 (c) shows the cumulative distribution of the number of complete peers in a network with 2000 peers. The results show that our Bee can efficiently diminish the maximum dissemination time for all network sizes, and the MDT of Bee is only 4218 seconds that is very close to the lower bound.

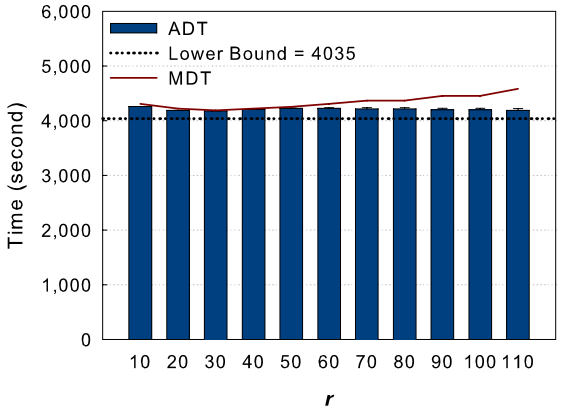


Fig. 4. Effect of various rates r on the ADT and MDT in a 1000 peers network.

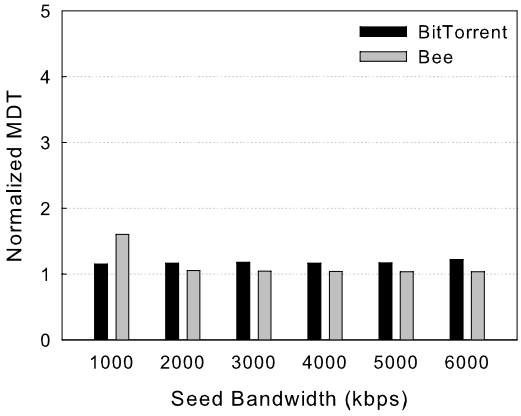


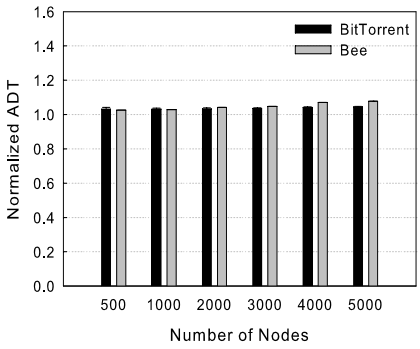
Fig. 5. Effect of various seed bandwidth on MDT in a network with 1000 peers.

C. Heterogeneous Environment

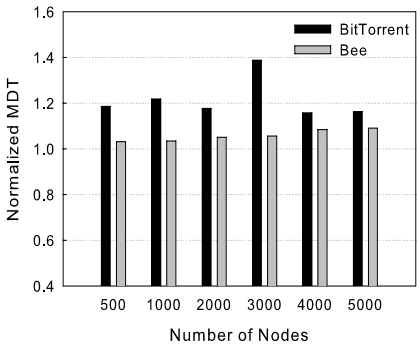
Next, we evaluate the performance of Bee and BitTorrent protocol in a heterogeneous network that consists of two types peers, one of which has a higher upload/download capacity (3000/1000 Kbps) than the other (1500/384 Kbps). As above simulations, all peers join the system at the initial stage, and leave when they finish downloading. In this section, we have evaluated all performance metrics of Bee but only show the impact of network size, because other results are similar to that in the homogeneous environment. In this scenario, the lower bound is 2333 seconds.

Fig. 7 (a) shows the ADT metircs for Bee and BitTorrent. It seems that both them are good on the ADT metric, because their ADTs approach the lower bound, but our Bee is better than BitTorrent in average. Besides, Fig. 7 (b) shows the normalized MDT metric for Bee and BitTorrent, and we can see that Bee is almost twice faster than the BitTorrent in the MDT metric. We also show the cumulative distribution of the number of complete peers in a network with 2000 peers in Fig. 7 (c). The result shows that a peer with higher capacity leaves faster than the peer with lower capacity in BitTorrent. After the peers with higher capacity leave system, the system capacity deceases significantly and the dissemination time of the peers with lower capacity will be prolonged. It fits our analysis in section II.

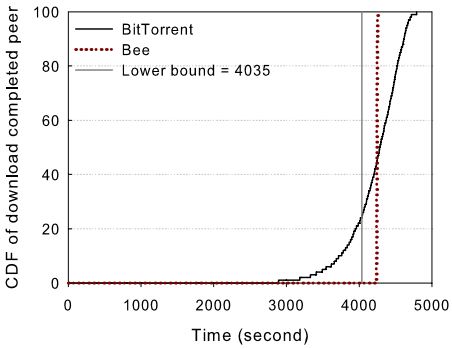
In order to analyze the difference between the performance of Bee and BitTorrent in a heterogeneous network in deep, we show the average uploading link utilization of peers, including the seed (6000 Kbps) and two type peers (1000 Kbps and 400 Kbps) in Fig. 8. We can easily to see that the upload link utilization of each peer is over 90%, which means that the overall upload capacity of the network is close to fully utilized. However, in BitTorrent, a peer with higher upload capacity should exchange block with another one with similar upload bandwidth, because the TFT peer selection strategy is likely to reward for the one with similar upload bandwidth. As a result, the overall upload link utilization of BitTorrent is efficient, but the design philosophy is based on egoist, so the lower capacity peers need more time to download data.



(a) Comparison of Bee to BitTorrent for ADT

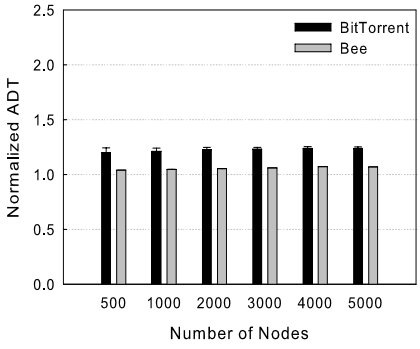


(b) Comparison of Bee to BitTorrent for MDT

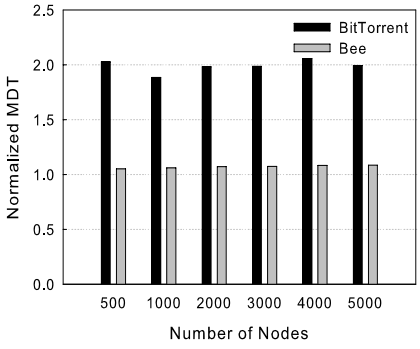


(c) Cumulative distribution of download peers on Bee and BitTorrent with 2000 nodes.

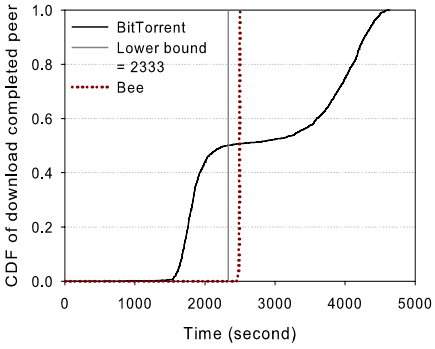
Fig. 6. Scalability comparison of Bee to BitTorrent in homogeneous environments.



(a) Comparison of Bee to BitTorrent for ADT



(b) Comparison of Bee to BitTorrent for MDT



(c) Cumulative distribution of download peers on Bee and BitTorrent with 2000 nodes.

Fig. 7. Scalability comparison of Bee to BitTorrent in heterogeneous environments.

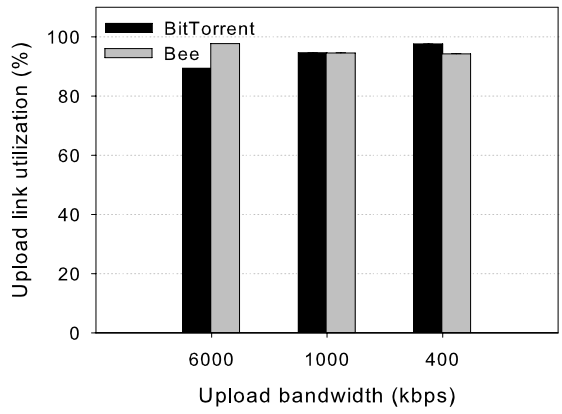


Fig. 8. The average uploading link utilization of peers, seed (6000Kbps), higher capacity peers (1000Kbps) and lower capacity peers (400Kbps)

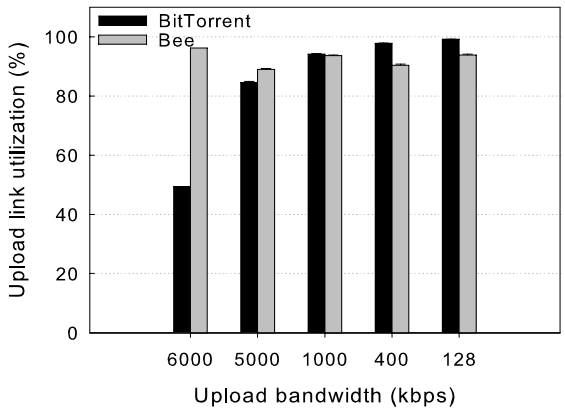
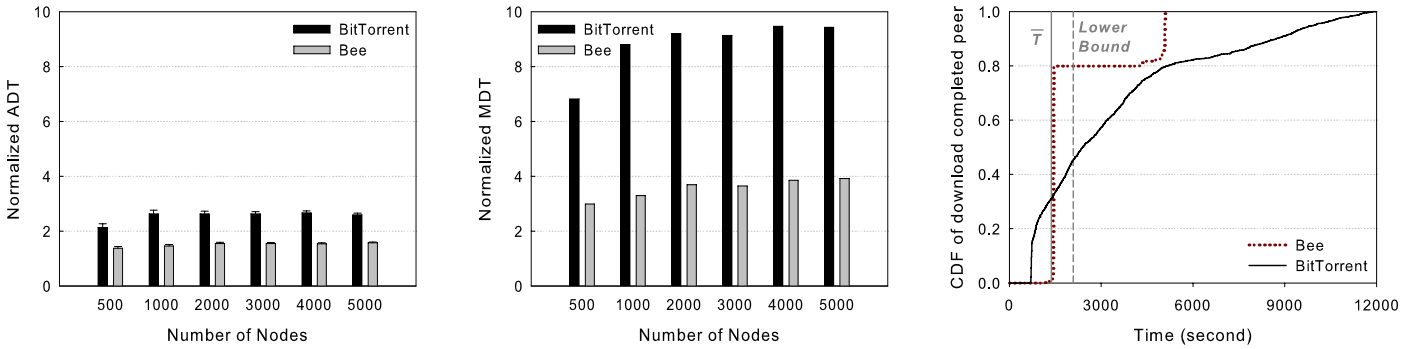


Fig. 9. The average uploading link utilization of peers, seed (6000Kbps), others indicate peer capacity

D. More Heterogeneous Environment

In this section, we repeated the same experiment in a more heterogeneous network with four types of peer capacity, which refers to the peer capacity distribution of realistic Gnutella system [32]. Actually, it presents a very complex network condition. As previous experiments, we calculate the lower bound of this scenario and it should be 2089 seconds ($\frac{size(F)}{\min\{D_i\}} = \frac{1638400}{784}$). Our first experiment is to study the impact of the number of peers from 500 to 5000, but all results are very similar to the results in the heterogeneous networks except for the seed bandwidth utilization.

Fig. 9 shows the upload link utilization of seed in Bee and



(a) Comparison of Bee to BitTorrent for ADT (b) Comparison of Bee to BitTorrent for MDT (c) Cumulative distribution of download peers on Bee and BitTorrent with 2000 peers.

Fig. 10. Scalability comparison of Bee to BitTorrent in more heterogeneous environments.

BitTorrent. It is easy to see that the upload link utilization of seed in Bee is 90%, but only 50% in BitTorrent system. From our viewpoints, when the variance of upload capacity increases, more peers with higher capacity will leave system early. When it happens, some of the seed's upload bandwidth may be idle, because the download bandwidth of the peer with lower capacity is too small to fully exploit the seed's upload bandwidth. As the result, the upload link utilization of seed decreases when the variance of upload capacity increases. In contrast, the upload link utilization of seed in Bee is the same regardless of the degree of upload capacity heterogeneity in the network.

We now explore the scalability of Bee in the complex network environment. As shown in Fig. 10 (a), Bee has a smaller ADT metric and the ADT is close to the lower bound. However, BitTorrent loses the ascendency in this case, each peer in BitTorrent needs double time to finish downloading. Fig. 10 (b) shows the normalized MDT metric of Bee and BitTorrent. This result also shows that Bee is almost twice faster than the BitTorrent in MDT metric. We also show the cumulative distribution of the number of complete peers in a network with 2000 peers in Fig. 10 (c). It is easy to see that 80% peers leave system at the $\overline{T}$ time (Recall that the Eq. (5) that we defined in section II.) and the remained 20% peers prolong the MDT of Bee system. In fact, these 80% peers are higher capacity peer, and the dissemination time of remained peers is limited by their download capacity. When the higher capacity peers leave system early, the service capacity of overall system will decease very significantly, and it results in a long dissemination time in both Bee and BitTorrent.

Fig. 11 shows the cumulative distribution of the number of complete peers with some modification. We extend the download capacity of poor peers to make sure that each peer can download a complete file before the lower bound. In fact, we increase the download capacity of the slowest peers from 784 Kbps to 1200 Kbps. Fig. 11 shows the CDFs of the download time for the two protocol in the case of a network with 2000 peers. The top one is the CDF with 784 Kbps (minimum download capacity) and the bottom is the CDF with 1200 Kbps. From the results in Fig. 11, we can observe that the

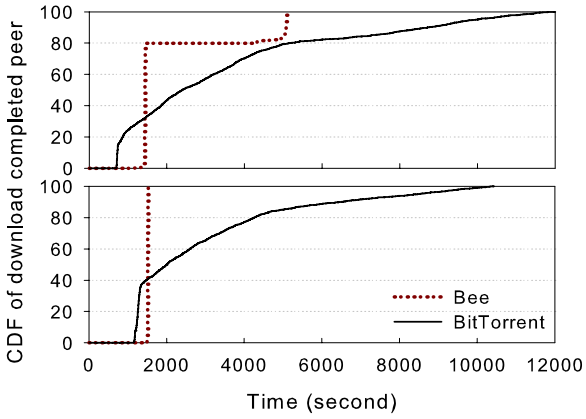


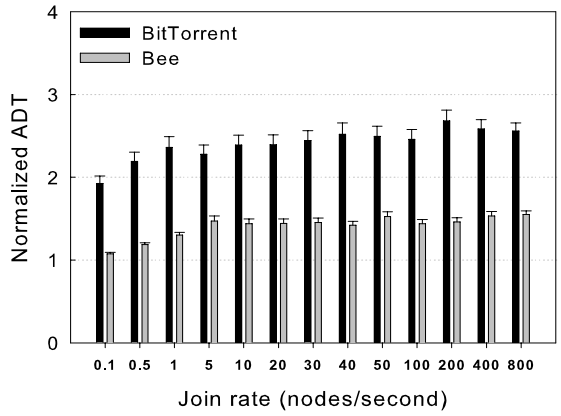
Fig. 11. Cumulative distribution of download peers on Bee that each peer can download a complete file before the lower bound.

dissemination time in Bee also approaches the lower bound, when the ($\frac{size(F)}{\min\{D_i\}} = 1365.3$) is smaller than ($\overline{T} = 1374.9$). The result implies that the performance of Bee is independent of the degree of network heterogeneity, and the dissemination time in Bee can approach the lower bound when there are no bottleneck links at the downstream peers.

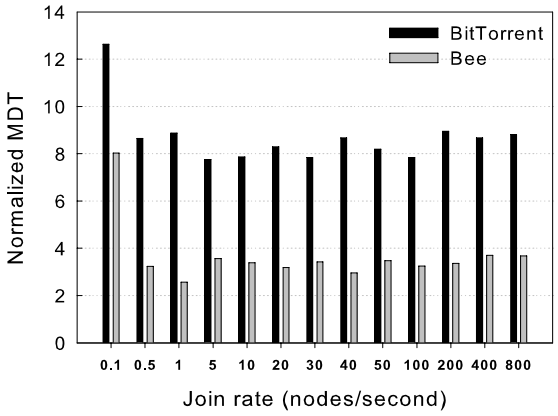
In sum, all peers in Bee are able to finish downloading very quickly even in complex network scenarios. On other hand, only the dissemination time of the peer with slowest downloading capacity can not approach the lower bound. However, that dissemination time of Bee will be significantly affected by the upload capacity of the seed and the peer with slowest download capacity, whereas in the Bee protocol the performance perceived by all peers will not change significantly.

E. The Effect of Join Pattern

In this section, we study the impact of different user arrival patterns on the performance of Bee and BitTorrent systems. In following experiments, we vary the peer join rate to verify the performance variation of Bee and BitTorrent systems. All



(a) Comparison of Bee to BitTorrent for ADT



(b) Comparison of Bee to BitTorrent for MDT

Fig. 12. Performance comparison of Bee to BitTorrent with increasing arrival rate in more heterogeneous environment.

experiments in this section use following two sets of settings: 1) a poisson arrival process with a total of 1000 peers in each set of simulation. 2) an arrival pattern derived from a tracker log of a Redhat 9 distribution torrent. And capacity of each peer is randomly selected from four types of peer capacity as shown in Table 1. The upload bandwidth of the seed is 6000 Kbps.

Fig. 12 shows the performance when using different user arrival rates for the Bee and BitTorrent system. As Fig. 12 (a) shows, we can see that the ADT linearly increases when the arrival rate increases both in Bee and BitTorrent. The reason for this result is that when more peers join at same time, more peers need to wait at the start-up time to receive the first block, so a larger arrival rate causes a longer ADT. Fig. 12 (b) shows the MDT metric of the Bee and BitTorrent. We can easily observe that when the arrival rate is low, the MDTs of Bee and BitTorrent both increase. The reason for this is that, when the arrival rate is low, the service capacity of overall system is also low. So peers in our Bee and BitTorrent need more time to finish downloading the file. However, our Bee outperforms BitTorrent in the both metrics.

We now evaluate Bee and BitTorrent in a realistic join pattern. In this experiment, each peer joins the system according to the tacker log of a a Redhat 9 distribution torrent, and all simulation settings are the same as that in above experiment. Note that peer capacity consists of four types as shown in Table I, so the lower bound in this case is that $\frac{size(F)}{\min\{D_i\}} = 2089$ seconds. Fig. 13 depicts the distribution of peer joining time, the log was collected over 12000 peers joining time. We can see that the flash crowd happens at the release time for a new version of Redhat ISO.

All the results are shown in Fig. 14. First, we demonstrated the download completion time of each peer for Bee and BitTorrent in Fig. 14 (a). It is clear that 83% peers in Bee finish the their download before 2000 seconds. On the contrary, only 50% peers can leave BitTorrent system at 2000 seconds. Moreover, Bee only needs 1/3 download time of BitTorrent to finish the file dissemination. In the following, we show the upload utilization of peers of Bee and BitTorrent in Fig. 14 (b) and (c). We can see that the upload utilization of each peer in Bee is almost fully utilized. And the BitTorrent has poor upload utilization in peers with higher capacity. One reason for this is that BitTorrent's random peer selection strategy maybe pair a higher capacity peer with a lower capacity peer. When that happens, the upload link bandwidth can not be fully utilized, because the download capacity of peer with lower capacity is too small or it doest have block to exchange with the higher capacity peers. Hence, the upload link utilization of peers with higher capacity may not be fully utilized in extreme heterogeneous networks.

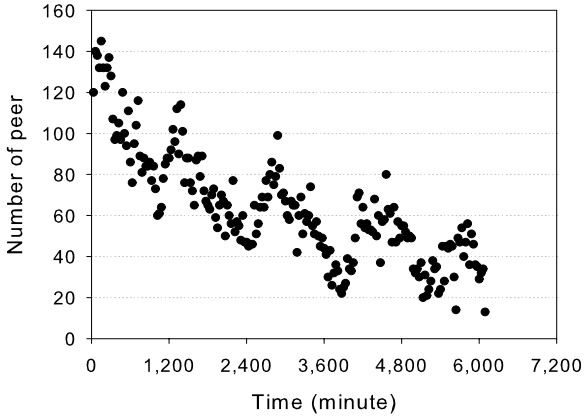
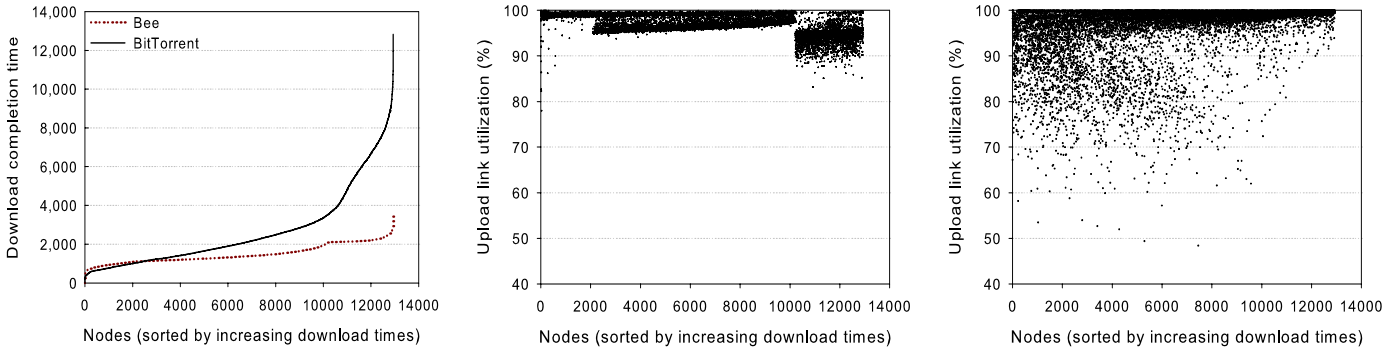


Fig. 13. The distribution of peer joining time.

VI. Related Work

In recent years, there are tremendous interests in building content delivery networks to address the data dissemination problem, which aims to deliver large-sized data to a group of nodes spread across a wide-area network. However, how to design an efficient protocol to achieve the lower bound of data dissemination time for the problem has not been discussed in depth in previous literatures. For the content



(a) Distribution of dissemination time (b) Upload link utilization of each Bee peer (c) Upload link utilization of each BitTorrent peer

Fig. 14. Performance of Bee and BitTorrent with arrival rate from Redhat 9 tracker log.

delivery networks, we divide existing approaches into three categories: (1) Multicast-based content delivery networks, (2) infrastructure-based content delivery networks, and (3) P2P-based (Swarming) content delivery networks. In addition, there are some advanced coding techniques [30-31] for content delivery, but we do not discuss that in this paper.

A. Multicast-based Content Delivery

A simple way to distribute content fast is to replace several unicast streams with one single multicast stream. This can be done in either the native IP multicast [3], or the application layer multicast. The native IP multicast approaches focused on constructing efficient network level multicast trees to deliver individual packets from the source to the receivers. Unfortunately, the native IP multicast approaches have some fundamental problems with reliability, congestion control, heterogeneity, and deployment.

As the IP mulitcast can not be widespread deployed, a large number of efforts then focused on building application-level overlays [23, 26] to construct and maintain a multicast tree using only the end peers. However, those tree-based techniques face several fundamental limitations on scalability and performance. For example, a single connection failed in a tree may significantly impact the throughput to all participated peers. Further, the peers only receive data from its parents in the tree, thus the failure of a single peer may impact overall system reliability. For many real situations, to construct a reliable and scalable multicast tree still has many difficult problems.

Because of the limitations for single overlay tree approach, recent studies (e.g., Bullet [6], Splitstream [7], Bullet' [18] and [27]) build multiple overlay mulitcast trees or overlay meshes to establish a more efficient and robust system where the departure of a peer only causes minor disruptions. Creating and maintaining multicast trees may provide an optimized architecture for the content delivery, however, the characteristics of churn effect in the P2P network may cause high maintenance cost for these approaches. Therefore, these architectures may not always be the best approaches.

B. Infrastructure-Based Content Delivery

Content Delivery Networks (CDNs) [29], such as Akamai Technologies[2], have been proposed to improve accessibility for the commercial companies. CDNs are dedicated collections of servers located strategically across the wide-area Internet. Content providers, such as multimedia video sources, contract with commercial CDNs to distribute content. CDNs are compelling to content providers because the responsibility for distributing content is offloaded to the CDN infrastructure. Many new infrastructures for the CDNs have recently been developed to focus on distributing large files, while related research systems include CoBlitz [17], Spider [28]. These systems offer a stable and performance-predictable content delivery architecture, especially for the businesses that want to offload their bandwidth but need to delivery a large content.

However, regardless of how many nodes in the CDNs are deployed, they are still limited in provisioned service capacity explicitly. When the demand of users grows too fast, the performance of these system may degrade significantly. The difference between these approaches and the Bee system is that the service capacity of Bee increases as the number of peer increases. Thus, we believe that the Bee system is able to handle the flash crowd user arrival pattern. In addition, we also believe that our Bee protocol can become a building block of CDNs by only minor modifications.

C. P2P-Based Content Delivery

The P2P file swarming technology has received a lot of attention from Internet users and networking researchers [4, 5, 8]. The main concept of the file swarming is inspired from the parallel-downloading mechanism [19]. For the file swarming technologies, we provide a summary as follows.

BitTorrent [4] is a very popular content distribution system which is successful for its efficiency in delivering a large file. There are two key mechanisms used in the BitTorrent system, namely, the "Tit-For-Tat" (TFT) peer selection policy and the local rarest first piece selection (LRF) strategy. The TFT peer selection policy aims to prevent the free-riders that refuse to contribute bandwidth to other peers. Another important feature

[2]See www.akamai.com.

of BitTorrent is the local rarest first piece selection (LRF) strategy. The objective of the LRF strategy is to enhance the piece availability of overall file. In general, BitTorrent is a very simple but efficient protocol for content delivery.

Slurpie [5] focuses on reducing load on servers and client download times when the number of downloading clients is tremendous. Slurpie uses an adaptive downloading mechanism which can improve client's performance according to its capacity, and adopts a random back-off algorithm to precisely control load on the server. However, the dissemination time of Slurpie may be prolonged, especially when the server has some rare blocks that are requested by a large number of clients, because the back-off algorithm prevents too many clients downloading from the server simultaneously.

Crew [8] is a new gossip-based protocol for data dissemination, and it performs a faster dissemination than the BitTorrent protocol in experiments. However, the overhead for gossiping arises form the redundant gossip messages. Crew needs a complex control mechanism to reduce the message overhead, because large message overhead may lead to decease total system throughput and slow down the dissemination time when a large number of recipients exist in the system. In fact, Crew is the fastest data dissemination protocol in their experiments [8], but the authors still do not show how close it approaches to the lower bound.

VII. Conclusion and future work

In this paper, we present an analysis of the lower bound of the dissemination time for the data dissemination problem. We also present the Bee protocol to capture the following notions: (i) all peers leave the system at the same time as much as possible, and (ii) each peer utilizes its upload bandwidth as much as possible. Note that, though one may argue that we could make the seed powerful enough to hold one last piece of the file until the best timing to upload it to the peers, so that all peers leave the system at the same time as much as possible. However, when the resources, such as bandwidth and computing power of peers often are heterogeneous and the user arrival pattern is dynamic, the computation complexity for the best timing is extremely high. On the other hand, our Bee protocol does not require any scheduling knowledge and each peer makes its own decision to download blocks only according to local information. We have conducted extensive simulations to evaluate the performance of the Bee protocol, and the simulation results show that the dissemination time in the Bee system can approach the lower bound when there are no bottleneck links at the server side or downstream peers. Specially, in the simulations on heterogeneous network environment presented in this paper, dissemination time of Bee is only 1/5 of that of BitTorrent, which is only 10% higher than the theoretical lower bound. As for the arrival traffic derived from a software release log, dissemination time of Bee is only 1/3 of that of BitTorrent system.

Currently, fast distribution of large scale software updates to millions of Internet users is becoming a critical task in today Internet, technology such as Bee protocol presented in this paper will thus play a major role in its future development. It is also interesting to look at how Bee protocol can be incorporated in P2P streaming applications.

Acknowledgment

The authors would like to thank Dr. Kuan-Ta Chen for comments on an earlier draft of this paper.

References

[1] B. Knutsson, H. Lu, W. Xu and B. Hopkins, "Peer-to-peer Support for Massively Multiplayer Games," *In Proceedings of IEEE INFOCOM*, Apr 2004.

[2] Christos Gkantsidis, Thomas Karagiannis, Pablo Rodriguez, and Milan Vojnovi, "Planet Scale Software Updates," *In Proceedings of ACM SIGCOMM*, Pisa Itlay 2006.

[3] Supratik Bhattacharyya, James F. Kurose, Donald F. Towsley, Ramesh Nagarajan, "Efficient rate-controlled bulk data transfer using multiple multicast groups," *IEEE/ACM Trans. Netw.*, 11(6): 895-907, 2003.

[4] B. Cohen, "Incentives build robustness in BitTorrent," *in Proceedings of first ACM workshop on Economics of Peer-to-Peer System*, Berkeley, USA, June 2003.

[5] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol," *In Proceedings of IEEE INFOCOM*, Apr 2004.

[6] Dejan Kosti, Adolfo Rodriguez, Jeannie Albrecht, Amin Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," *In Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003.

[7] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," *In Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003.

[8] Mayur Deshpande, Bo Xing, Iosif Lazardis, Bijit Hore, Nalini Venkatasubramanian, Sharad Mehrotra, "Crew: A Gossip-based Flash-Dissemination System," *In Proceedings of IEEE ICDCS*, 2006.

[9] X. Yang and G. de Veciana, "Service Capacity of Peer to Peer Networks," *in Proceedings of IEEE INFOCOM*, Apr, 2004.

[10] D. Qiu and R. Srikant, "Modeling and performance analysis of Bittorrent-like peer-to-peer networks," *in Proceedings of ACM Sigcomm*, Portland, OR, USA, 2004.

[11] T. Karagiannis, A. Broido, M. Faloutsos, Kc claffy, "Transport Layer Identification of P2P Traffic," *in Proceedings of ACM IMC*, Taormina, Sicily, Italy, October 25-27, 2004.

[12] A. M. Farley, "Broadcast Time in Communication Networks," *SIAM Journal on Applied Mathematics*, 39(2):385–390, 1980.

[13] K HULLER , S., AND K IM , Y.-A, "On broadcasting in heterogeneous networks," *in Proceedings of ACM-SIAM Symposium on Discrete algorithms*, 2004.

[14] Mayur Deshpande, Nalini Venkatasubramanian And Sharad Mehrotra, "Heuristics for Flash-Dissemination in Heterogenous Networks," *in Proceedings of International Conference on High Performance Computing (HiPC)*, 2006.

[15] M. Izal, G. Urvoy-Keller, E.W. Biersack, P. Felber, A. Al Hamra, and L. Garces-Erice, "Dissecting BitTorrent: Five Months in a Torrent.s Lifetime," *In Proc. Passive and Active Measurement*, April 2004.

[16] A. Bharambe, C. Herley, and VN Padmanabhan, "Analyzing and Improving BitTorrent Performance," *in Proceedings of IEEE INFOCOM*, Barcelona, Spain, Apr. 2006.

[17] KyoungSoo Park, Vivek S. Pai, "Scale and Performance in the CoBlitz Large-File Distribution Service," *in Proceedings of Third Symposium on Networked Systems Design and Implementation*, San Jose, CA, May 2006.

[18] Dejan Kostic, Ryan Braud, Charles Killian, Erik VandeKieft, James W. Anderson, Alex C. Snoeren, "Maintaining High Bandwidth under Dynamic Network Conditions," *in Proceedings of the USENIX Annual Technical Conference*, Anaheim, CA, April 2005.

[19] A. Kirpal, P. Rodriguez, E.W. Biersack, "Parallel-Access for Mirror Sites in the Internet," *in Proceedings of IEEE INFOCOM*, Tel-Aviv, Apr. 2000.

[20] Robert L. Carter, Mark E. Crovella, "Server Selection Using Dynamic Path Characterization in Wide Area Networks," *in Proceedings of IEEE INFOCOM*, 1997.

[21] Jacob Strauss, Dina Katabi, and Frans Kaashoek, "A Measurement Study of Available Bandwidth Estimation Tools," *In the Proceedings of the ACM Internet Measurement Conference*, Miami, Florida, October 2003.

[22] Aditya Akella, Srinivasan Seshan and Anees Shaikh, "An Empirical Evaluation of Wide-Area Internet Bottlenecks," *in Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference*, Miami, Florida, 2003.
[23] Yang-Hua Chu, Sanjay Rao, and Hui Zhang, "A case for end system multicast," *in Proceedings of ACM Sigmetrics*, Santa Clara, CA, 2000.
[24] Gang Wu, Tzi-cker Chiueh, "How Efficient is BitTorrent?," *in Proceedings of ACM MMCN*, Jan 2006.
[25] Dave Levin, Rob Sherwood, Bobby Bhattacharjee, "Fair File Swarming with FOX," *in Proceedings of International Workshop on Peer-to-Peer Systems*, Febuary 27-28, 2006.
[26] S. Banerjee, B. Bhattacharjee, and C. Kommreddy, "Scalable Application Layer Multicast," *in Proceedings of ACM SIGCOMM*, 2002.
[27] Z. Xiang, Q. Zhang, W. Zhu, Z. Zhang, and Y-Q. Zhang, "Peer-to-peer based Multimedia Distribution Service," *IEEE Transactions on Multimedia*, Volume 6, p.343-355, April 2004.
[28] Samrat Ganguly, Akhilesh Saxena, Sudeept Bhatnagar, Suman Banerjee, Rauf Izmailov, "Fast Replication in Content Distribution Overlays," *in Proceedings of IEEE INFOCOM*, April 2005.
[29] G. Pallis and A. Vakali, "Insight and Perspectives for Content Delivery Networks," *Communications of the ACM*, vol. 49, no. 1, ACM Press, NY, USA, pp. 101-106, January 2006.
[30] John W. Byers , Jeffrey Considine , Michael Mitzenmacher , Stanislav Rost, "Informed content delivery across adaptive overlay networks," *IEEE/ACM Transactions on Networking (TON)*, v.12 n.5, p.767-780, October 2004.
[31] C. Gkantsidis, P. Rodriguez, "Network Coding for Large Scale Content Distribution," *in Proceedings of IEEE INFOCOM*, Miami, March 2005.
[32] Stefan Saroiu, P. Krishna Gummadi, Steven D, "A Measurement Study of Peer-to-Peer File Sharing Systems," *in Proceedings of ACM MMCN*, Jan 2002.