On Optimal File Distribution in Practical Mesh-Based Overlay Networks

Xiao Su*, Yan Bai+, Suchreet K. Dhaliwal*
*Department of Computer Engineering, San Jose State University, San Jose, USA
+Institute of Technology, University of Washington Tacoma, Tacoma, USA
Email: xiao.su@sjsu.edu, yanb@u.washington.edu, suchreetkaur@gmail.com

***Abstract*— Distributing large video files or operating system images over the Internet requires file servers with high bandwidth and large storage capacity. Overlay networks, including content distribution networks (CDN) and peer-to-peer (P2P) systems, are promising network models for large file distributions. Both CDN and P2P leverage bandwidth and storage resources between content distribution servers and individual nodes, so that they can scale to a larger number of nodes easily. Previous work on large file distribution mainly focused on minimizing the distribution time of a fully connected overlay network. In a fully connected overlay network, each individual node is connected to every other node in the network. However, most practical CDN and P2P systems are based on a partially connected mesh topology, where nodes are typically connected to a subset of other nodes. In this paper, the distribution time of practical mesh-based overlay systems is analyzed and a lower bound on the file distribution time is established. Our algorithms consist of two steps. First, we decompose the mesh network into multiple spanning trees so that the load on each node is balanced. We show that the construction of balanced spanning trees is NP-complete and propose a few heuristics to tackle it. The second step, we derive the optimal system distribution time based on the multiple spanning tree topology, node bandwidths and file size. In this step, an optimal file segmentation algorithm is developed, in which a file is divided into unequal-sized pieces and allocated to individual nodes based on the available bandwidth. We validate our theoretical analysis via experiments and investigate how system design parameters, such as node churning and implementation complexity, affect system distribution time.**

***Index Terms*— content distribution networks (CDN), peer-to-peer networks (P2P), mesh network, file distribution, overlay networks.**

I. INTRODUCTION

As network bandwidth increases, the amount of large files, such as video files and operating system updates, distributed over the Internet has increased tremendously. Traditional client-server architecture faces challenges for large file distribution over the Internet. In the client-server architecture, a server stores all the files and delivers them to the clients. Such a system is susceptible to flash crowd as the number of clients increases rapidly.

To alleviate overloading on an original content server, two types of systems are deployed in practice: server-assisted systems and peer-assisted systems.

In a server-assisted system, dedicated proxy or replication servers are installed at the edge of a network. In most cases, the proxy or replication server close to the clients provides a quick response to the clients, and therefore, the burden on the original servers is reduced. Two main commercial server-assisted systems are pull-based web cache and push-based content distribution network (CDN). In cache-based solutions, only the first file request is passed to the original content server. Once the file has been "pulled" to the cache, subsequent requests for the same file are handled by the proxy cache. CDNs consist of a large number of replication servers that host files. The files are proactively "pushed" to the replication servers by original content servers. These replication servers are owned and deployed by CDN companies, such as Akamai, Limelight, and Amazon CloudFront. The client request for a file is "redirected" to the closest CDN server by a smart DNS remapping.

Compared to the organized nature of server-assisted solutions, peer-assisted architecture leverages the computing and networking resources of every participating node connected autonomously. No proxy or edge servers are required.

The first peer-assisted system is developed by Napster. Napster improves client-server systems by allowing file sharing between peers. In Napster, popular files are fully distributed throughput the network. In other words, a peer can obtain such files from different sources, thus largely increasing the availability of content. Napster separates file-sharing process into two phases: file search and file transfer. In Napster, file search uses a client-server model, while file transfer uses a peer-to-peer model.

Following Napster, many other P2P file sharing protocols such as Gnutella, Freenet, FastTrack, KaZaA and BitTorrent [1] are developed. Among them BitTorrent [2] quickly gained popularity and became very successful. BitTorrent systems consist of two types of peers: leechers and seeders. Leechers are the peers who are in the process of downloading file pieces, while seeders are the peers who have already downloaded the whole file. Both Leechers and seeders build up a mesh-based peer-to-peer overlay network based on data requests.

Unlike other P2P systems, BitTorrent includes many unique features. First of all, it provides a number of incentive mechanisms such as tit-for-tat to fight against

Manuscript received February 5, 2010; revised June 25, 2010; accepted July 27, 2010.

free-riding: a peer is not allowed to only download files without uploading any files [3].

Second, BitTorrent splits a file into equal-sized pieces, when the file is distributed. Each piece carries a cryptographic hash-based checksum. A user can verify if the requested piece is correctly downloaded.

Third, BitTorrent implements a piece selection strategy called rarest-piece-first algorithm. According to the algorithm, a BitTorrent user downloads the file pieces that are held by the least number of peers in its peer list. The algorithm, therefore, promotes even distribution of file pieces to available peers. However, for a BitTorrent user, its rarest piece is calculated based on the piece numbers provided by the peers in the user's peer list. It will not be the same piece if calculated from the piece numbers provided by all the peers in the system. This "local" view of file piece availability makes it possible for some file pieces to be held by more peers than other pieces.

Peer-assisted file transfer and server-assisted CDNs have gained huge popularity in practice. A key research question is: can an optimal performance be achieved for all the peers or servers in a mesh-based overlay network? In this paper, we measure performance using system distribution time, which is defined as the time for every node in a system to receive a complete copy of a file. In order to answer the above question, we investigate the following fundamental issues on mesh-based overlay networks.

- What is the optimal system distribution time when file pieces are assigned to different nodes without duplication? Clearly, the study of this question would provide a lower bound on practical mesh-based file distribution since piece redundancy in an overlay network will increase system transfer time.
- How shall files be segmented into pieces to achieve the optimal distribution bound? Is equal size-based file split an optimal way?
- How to deal with peer churning and how does peer dynamics impact the optimal system distribution time?

The rest of the paper is organized as follows. Section II discusses the related work. Section III describes our P2P file distribution model. In Section IV, we analyze the minimum system distribution time and optimal file segmentation scheme. Section V investigates the impact of peer churning on system performance. Section VI proposes a decentralized node forwarding algorithm to reduce the complexity and states of nodes . Section VII validates our analysis and evaluates the proposed algorithms through extensive simulation experiments. Finallly, Section VIII summarizes the work in this paper and describes future work.

II. Related Work

Peer-to-peer file sharing performance has been extensively evaluated through measurement studies [4]–[7]. The free-riding problem first discussed in [7] has motivated many incentive schemes for P2P applications [8]–[11]. Besides free-riding, measurement studies have also discovered other limitations of P2P systems. Guo et. al. performed traffic analysis and found that in a single torrent, service availability decreases rapidly as peer arrival rate drops exponentially. Moreover, peer performance fluctuates with the torrent size [12]. These observations motivate the development of multi-torrent systems and encourage collaboration among multiple torrents.

There are some other works on performance modeling and analysis of BitTorrent systems. Qiu and Srikant use a deterministic fluid model to study how the number of seeders and leechers evolves over time, as a function of peer arrival and departure rates and uploading bandwidth. They have also analyzed average downloading time, based on the above three network parameters [13]. Ramachandran and Sikdar evaluate the average file downloading time for a participant in the P2P network by many other parameters, including file search time, queueing delays at network routers, number of simultaneous downloads at an individual peer, and number of available copies of a particular file. File downloading time is modeled from an end user's perspective [14]. Both modeling work in [13], [14] does not optimize the system distribution time.

There are some work on file distribution in CDNs. For example, FastReplica [15] is an algorithm to replicate a large file from a source node to CDN nodes. For a small group of CDNs consisting of k nodes, it divides the large file into k equal-sized subfiles and hands each subfile to a CDN node, which subsequently forwards the subfile to the remaining $k-1$ nodes. FastReplica has been combined with application layer multicast to reduce its file distribution time [16]. However, FastReplica does not optimize the overall system distribution time. Moreover, it assumes a complete connection topology in which nodes are fully connected to each other.

Previous work that explicitly addresses the minimum system distribution time include the fluid-based analysis by Kumar and Ross [17] and by Mundinger et. al. [18]. In [17], the minimum system distribution time is derived using a bit-based fluid model and expressed as a function of file size, peers' uploading and downloading bandwidths. In [18], the optimal time to distribute a file of M chunks to N end users with uniform bandwidth is studied. The analysis is chunk-based. It assumes that the downloading bandwidth is infinite. Both analyses use a network topology of a complete graph, where every node is connected to each other. The complete topology is difficult to achieve in a large-scale distributed environment, such as the Internet.

Chan et. al. develop several scheduling algorithms to decide the piece request sequences and their corresponding supplying peers. They also study minimizing system distribution time, when files are divided into equal-sized pieces [19].

To authors' knowledge, none of the work discussed in the above investigates the case where a file is divided into pieces with unequal size. In this paper, we propose an algorithm to optimally divide a file into unequal-sized pieces to minimize the system distribution time. Furthermore, Our work analyzes the minimum system distribution time using a realistic mesh-based topology, usually found in BitTorrent-like P2P systems, where every node is not connected to each other.

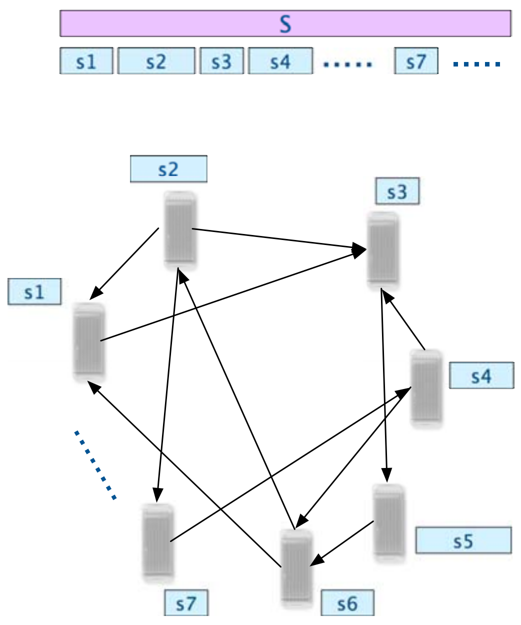


Figure 1. A file distribution network.

III. File Distribution Model

In a file distribution network, n nodes are connected in a mesh. As illustrated in Fig. 1, a content server divides an original file S into n segments, $s_1, s_2, \cdots, s_n$, and assigns each of these segments to a different node. Each node is then responsible for propagating its assigned segment to the rest of nodes in the network. This distribution model is different from BitTorrent in two aspects. First, there is no duplication of file segments in different nodes, and each node has a unique segment. Second, file segments are "pushed out" from the source node, rather than "pulled in" by a requesting node. This forces each node to fully utilize its uploading bandwidth and make a significant contribution to file distribution process. By removing segment redudancy in different nodes and fully utilizing nodes' uploading bandwidth, we achieve a lower bound on the file distribution time of a practical mesh-based system.

In this paper, system distribution time is used as one main performance metric. It is defined as the time for every node in a system to receive a complete copy of a file.

IV. Optimal File Distribution in Mesh-Based Networks

We explain the file distribution process through a simple example. Fig. 2 shows a mesh network consisting of 4 nodes with their connections, uploading and downloading bandwidths. When a node is assigned to

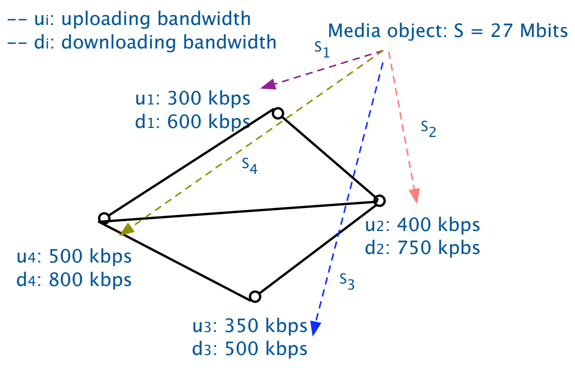


Figure 2. A non-tree topology with 4 nodes and their uploading and downloading bandwidths.

distribute its segment to every other node in the network, its distribution paths make a tree, including every node in an acyclic way. To ensure that the file segment originated at a node traverses the least network distance to reach all the other nodes in the network, a shortest path spanning tree (SPST) rooted at the node needs to be constructed.

Therefore, in a network of n nodes, the file forwarding paths constitute n shortest path spanning trees. In a given SPST, a node could be a leaf node, where it does not need to forward the segment any further, whereas a node may be s a root or an internal node, responsible for forwarding the segment down the distribution tree. To find the completion time of a node, we need to calculate its forwarding time as a non-leaf node in each SPST. An example SPST construction is shown in Fig. 3, where node 2 is responsible for forwarding segment s_1 in the SPST rooted at 1, and forwarding s_2 in the SPST rooted at node 2 itself. It does not need to forward any segment in SPSTs rooted at 3 and 4. Let us define the system distribution time as

$$T = \max\{t_1, t_2, \cdots, t_n\}$$

where t_i is the completion time of node i that includes the time to download its assigned segment, s_i, from the server and the time to forward segments in SPSTs as a non-leaf node.

For example, if the original file is divided into 4 equal-sized segments, *i.e.*, $s1 = s2 = s3 = s4 = 6750$ kbits, then it takes $T = 67.5$ seconds for every node to receive all the file segments. T is calculated as follows.

$$\begin{aligned} T = \max\{&\frac{6750}{600} + 2 \times \frac{6750}{300}, \frac{6750}{750} + 3 \times \frac{6750}{400} + \frac{6750}{400}, \\ &\frac{6750}{500} + 2 \times \frac{6750}{350}, \frac{6750}{800} + 3 \times \frac{6750}{500} + \frac{6750}{500}\} \\ = &\max\{56.25, 76.5, 52.07, 63.44\} = 76.5s \end{aligned} \quad (1)$$

Two questions arise when we closely inspect the example. First, as in Fig. 3, there are multiple ways to constructed a SPST rooted at a particular node. For example, an alternative SPST rooted at node 1 could reach node 3 through node 4, instead of node 2. The question is how we should construct the n SPSTs so that the distribution load is balanced for all the n participating nodes? Second, given SPSTs in place, how to divide a file

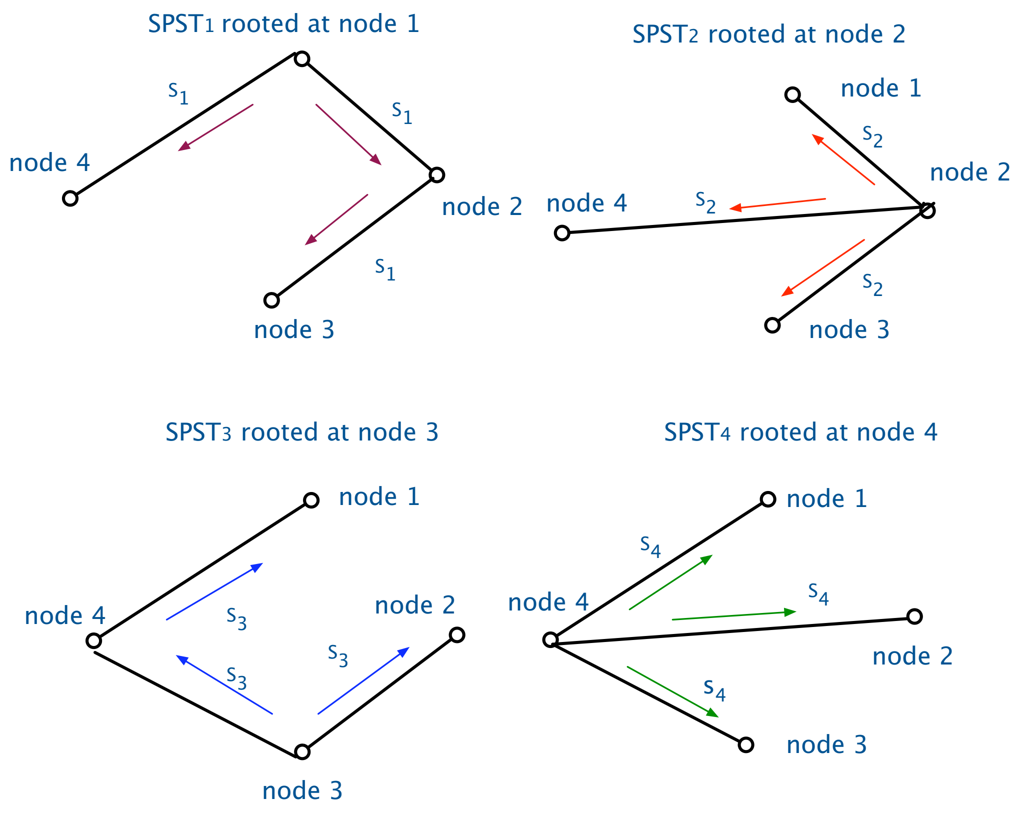


Figure 3. An example construction of shortest path spanning trees.

S into $s_1, s_2, \cdots, s_n$ to minimize the system distribution time? We address these two questions in the rest of this section.

A. Spanning Tree Constructions

Let us model a mesh network with n nodes as a graph $G = (V, E)$, where $n = |V|$. The challenge is to find n spanning trees so that the load for each node in the network is "balanced." The distribution load of a node can be estimated by its sum of outgoing degrees in these n SPSTs. Therefore, our problem reduces to collaboratively constructing n SPSTs to balance every node's sum of outgoing degrees in these n trees. We prove, In the following, that the problem of constructing balanced spanning trees is NP-complete by reducing it to a known NP-complete problem – degree constrained spanning tree problem [20].

Theorem 1: Constructing M balanced spanning trees is NP-complete.

PROOF: It is well known that changing an optimization problem into a search problem doesn't change its difficulty level. Hence, we rephrase this optimization problem as the following search problem: given graph $G = (V, E)$ and positive integers M and L, are there M spanning trees derived from G so that every node's sum of outgoing degrees in these M spanning trees is not larger than L?

First, it is easy to see that constructing M balanced spanning trees is in NP, since a nondeterministic algorithm can randomly pick M spanning trees and check in polynomial time whether the sum of every node's outgoing degrees in these M trees is smaller than L or not.

Second, let us reduce the degree constrained spanning tree problem to M balanced spanning tree problem. The degree constrained spanning tree problem is defined as follows. Given a graph $G = (V, E)$ and a positive integer $K \leq |V|$, the question is whether there is a spanning tree for G in which no vertex has degree larger than K. By setting M to 1 and K to L, the problem of finding a degree constrained spanning tree for $G = (V, E)$ in which no vertex has degree larger than K can be solved as a special instance of M balanced spanning tree problem.

As finding a degree constrained spanning tree is NP-complete, constructing M balanced spanning tree problem is also NP- complete. ■

Intuitively, constructing "balanced" SPSTs is hard due to two factors. First, there are multiple SPSTs rooted at some nodes, thus one challenge is how to select a SPST from these choices. Second, the order of choosing the root nodes of the n SPSTs results in different load allocation to nodes. Exhaustive enumeration needs to test $n!$ different orders of choosing root nodes.

We, therefore, consider three heuristic algorithms to build balanced SPSTs, and all of them apply Dijkstra's algorithm to generate an SPST rooted at a single node. The Dijkstra's algorithm maintains a distance value for each node. Initially, the source node has a distance zero, and all the other nodes have distance of infinity. At each step, it chooses the node with shortest distance as the next node to be marked permanent and updates the distance values of all its neighbors. A permanent node is the one whose shortest distance from the source node has been determined. The algorithm proceeds until all nodes have been marked permanent.

```
1. Randomly order and number the nodes from 1 to $n$
2. **for** each node $i$ from 1 to $n$ **do**
3.     using Dijkstra's algorithm to build a SPST
       rooted at node $i$.
4          at each iteration, when multiple nodes have
           the same distance, randomly choose a node
           to label permanent.
5. **end-for**
```

Figure 4. A random algorithm to construct n SPSTs.

```
1. Randomly order and number the nodes from 1 to $n$
2. initialize bal[$i$] to zero, for node $i$ from 1 to $n$.
3. **for** each node $i$ from 1 to $n$ **do**
4.     using modified Dijkstra's algorithm to build
       SPST rooted at node $i$.
5          at each iteration, when multiple nodes have
           the same distance, choose a node with
           minimum bal to label permanent.
6.     **for** each internal node $k$ of SPST rooted at $i$ **do**
7.         increase bal[$k$] by the sum of its outgoing
           degrees in the SPST rooted at $i$.
8.     **end-for**
9. **end-for**
```

Figure 5. A balanced algorithm to construct n SPSTs.

```
1. Number the nodes from 1 to $n$ in an increasing order
       of uploading bandwidth per connection
2. Follow steps 2 to 8 in Balanced Algorithm.
```

Figure 6. A ordered balanced algorithm to construct n SPSTs.

The first algorithm (Rand-SPST) is based on a random ordering of source nodes to build SPSTs. Moreover, it chooses a random node as the next permanent node, when multiple nodes have the same distance. This algorithm is simple, but is likely to result in uneven forwarding load at nodes. The steps are outlined in Fig. 4.

The second algorithm (Balanced-SPST) keeps track of outgoing degree sums of every node in SPST constructions, in order to balance the load of every node. Like the Rand-SPST algorithm, it randomly selects a node to be the root node of the next SPST. The steps are shown in Fig. 5.

The third algorithm (Ordered-SPST) arranges the nodes in an increasing order of its uploading bandwidth per connection. This is motivated by the observation that the later a node is chosen as a root node in the algorithm, the more likely it has a large sum of outgoing degrees, *i*.e., large distribution load. Therefore, putting more load on high capacity nodes instead of slower ones helps reduce the overall system distribution time. Similar to the Balanced-SPST algorithm, Ordered-SPST balances the number of outgoing degrees for all of the nodes. The steps are listed in Fig. 6.

B. Optimizing System Distribution Time

Once SPSTs have been built, how to divide a file into smaller segments and assign them to the nodes need to be determined. We formulate the segment assignment problem as a constrained optimization problem to minimize the system distribution time. Let u_i represent node i's uploading bandwidth, d_i its downloading bandwidth, and p_i^k as node k's number of forwarding paths in tree i. If k is a leaf node in tree i, then p_i^k is equal to 0. Given the above notations, the general optimization problem is formulated as follows.

$$\begin{aligned} \text{min} \quad & \max\{t_k, k = 1, 2, \cdots, n\} \\ \text{s.t.} \quad & t_k = \frac{s_k}{d_k} + \sum_{\text{tree } i=1}^{n} p_i^k \times \frac{s_i}{u_k}, \quad \text{for } k = 1, 2, \cdots, n \\ & \sum_{k=1}^{n} s_k = S \\ & s_k \geq 0, \quad k = 1, 2, \cdots, n \end{aligned} \tag{2}$$

It is easy to see that Eq. (2) is essentially a linear programming formulation, when we rewrite it as follows.

$$\begin{aligned} \text{min} \quad & T \\ \text{s.t.} \quad & \frac{s_k}{d_k} + \sum_{\text{tree } i=1}^{n} p_i^k \times \frac{s_i}{u_k} \leq T, \quad \text{for } k = 1, 2, \cdots, n \\ & \sum_{k=1}^{n} s_k = S \\ & s_k \geq 0, \quad k = 1, 2, \cdots, n \end{aligned} \tag{3}$$

Eq. (3) can be efficiently solved by any linear programming solver, such as *lp_solve* [21]. When we apply Eq. (3) to the example shown in Fig. 2, we get the following optimization problem.

$$\begin{aligned} \text{min} \quad & T \\ \text{s.t.} \quad & s_1/600 + 2s_1/300 \leq T \\ & s_2/750 + 3s_2/400 + s_1/400 \leq T \\ & s_3/500 + 2s_3/350 \leq T \\ & s_4/800 + 3s_4/500 + s_3/500 \leq T \\ & s_1 + s_2 + s_3 + s_4 = 27000 \\ & s_1, s_2, s_3, s_4 \geq 0 \end{aligned}$$

By solving the above problem, we have $(s_1, s_2, s_3, s_4) = (7516.6, 4963.79, 8119.78, 6399.83)$ kbits. The resulted system distribution time T is 62.6 seconds, clearly shorter than the distribution time of 76.5 seconds when we assign equal-sized segments to the nodes, as in Eq. (1).

V. Node Churning

In a dynamic overlay network, nodes are expected to join and leave frequently. Let us discuss the effect of node churning on the system. We assume that time is divided into time slots and a node joins at the beginning of a time slot and can leave at any time. When a node leaves the system, he leaves behind two types of unfinished media segments: one originating from itself and the other being forwarded by the node as an internal node. In effect, we can treat these unfinished segments as a new object to be delivered by the remaining nodes in the next time slot.

At the end of transmission slot 1, every node has a record of missing segments. When a departure node, i, is a forwarding node in a spanning tree rooted by k, then nodes in the upstream of the spanning tree have already received s_k. There is no need to request s_k from the original server. We call such segments partially distributed segments. For such a segment s_k, there are a set of supplying nodes that already have s_k and a set of demanding nodes that are waiting for s_k. We need to set up forwarding paths from the set of supplying nodes to the set of demanding nodes. In the process, we need to minimize the total number of forwarding edges. We borrow part of Dijkstra's algorithm for this purpose: initializing the permanently marked set to contain all the supplying nodes and repeating this process for every partially distributed segment. Once this step is completed, for each node i, we will calculate the total size of such partially distributed segments to be forwarded by node i, mc_i, $i = 1, 2, \cdots, m$, where m is the number of remaining nodes.

For segments that are needed by all the nodes, we request the server to divide them and re-distribute to the remaining nodes. The server needs to solve a linear programming problem to find out the distribution size for every remaining node, $sm_k, k = 1, 2, \cdots, m$. In this problem, the size of the media object is the sum of all missing segments, SM. Recall that mc_k is the size of partially distributed segments to be forwarded by node k, and we assume that node k sends mc_k to the original server. Given u_k, d_k, mc_k and SM, the server solves sm_k from the following optimization problem.

$$
\begin{aligned}
\min\ & T \qquad (4)\\
\text{s.t.}\ & \frac{mc_k}{u_k} + \frac{sm_k}{d_k} + \sum_{\text{tree } i=1}^{m} p_i^k \times \frac{sm_i}{u_k} \le T,\\
& \qquad \text{for } k = 1, 2, \cdots, m\\
& \sum_{k=1}^{m} sm_k = SM\\
& sm_k \ge 0, \quad \text{for } k = 1, 2, \cdots, m
\end{aligned}
$$

VI. Decentralized Reverse Path Forwarding

The server solves the optimization problems defined in Eq. (3) and Eq. (4) to find an optimal media allocation scheme, s_i, for $i = 1, 2, \cdots, n$. The server keeps the topology of an overlay network, so that it can calculate spanning trees based on its complete information about the overlay network. Every node needs to keep the following states for every spanning tree: whether it is an internal node, and if so, its descendant nodes in the tree. To communicate the state information from the server to every node consumes additional bandwidth and makes the server the only bottleneck in the system. Therefore, the solution of Eq. (3) serves as a lower bound on the system distribution time of a mesh-based overlay network. However, the deployment of this centralized algorithm is not scalable.

In practice, we can decentralize the algorithm by leveraging reverse path forwarding (RPF) algorithm [22], a widely used multicast routing protocol, to avoid forwarding loops. Every node uses RPF to decide whether to forward a segment received from a source node i. If node k receives a segment from a link that is on node k's shortest path to the source node i, then it knows that the segment arrives to him via the shortest path spanning tree. He will flood it to the rest of his outgoing links. Otherwise, node k will stop forwarding the segment. When every node implements RPF algorithm, a mesh-based overlay network is inherently decomposed into shortest path spanning subgraphs originated at each node in the network. The subgraphs are no longer trees, because there are cycles in the distribution paths.

To implement RPF, every node needs to know the shortest path to every other node in the network and the outgoing link for each shortest path. This is the well-known all-pairs shortest path problem, that underlies the construction of Internet routing tables. Applying Floyd-Warshall algorithm [23], we can solve the problem in $O(n^3)$, where n is the number of nodes in the network.

Fig. 7 outlines the RPF algorithm implemented by every node in the system. Fig. 8 shows the forwarding graphs, when RPF is applied by each node. The dashed edges are non-SPST edges and the dashed arrows represent the "redundant" packets that will be silently discarded in the network. By sending "redundant" packets and costing additional bandwidth, every node can independently decide where to forward packets. It does not rely on the server to transmit its forwarding path in every spanning tree.

```
for every node $k$ in the system do
    set seen[k] to 0
end-for
upon receiving segment $s_a$
    if $s_a$ is received along a path in SPST rooted at
    node $a$
        1. forward it to every outgoing link except
        the one $s_a$ is received;
        2. set seen[a] to 1
    else
        don't forward $s_a$
    end-if
```

Figure 7. Reverse path forwarding algorithm implemented at every node.

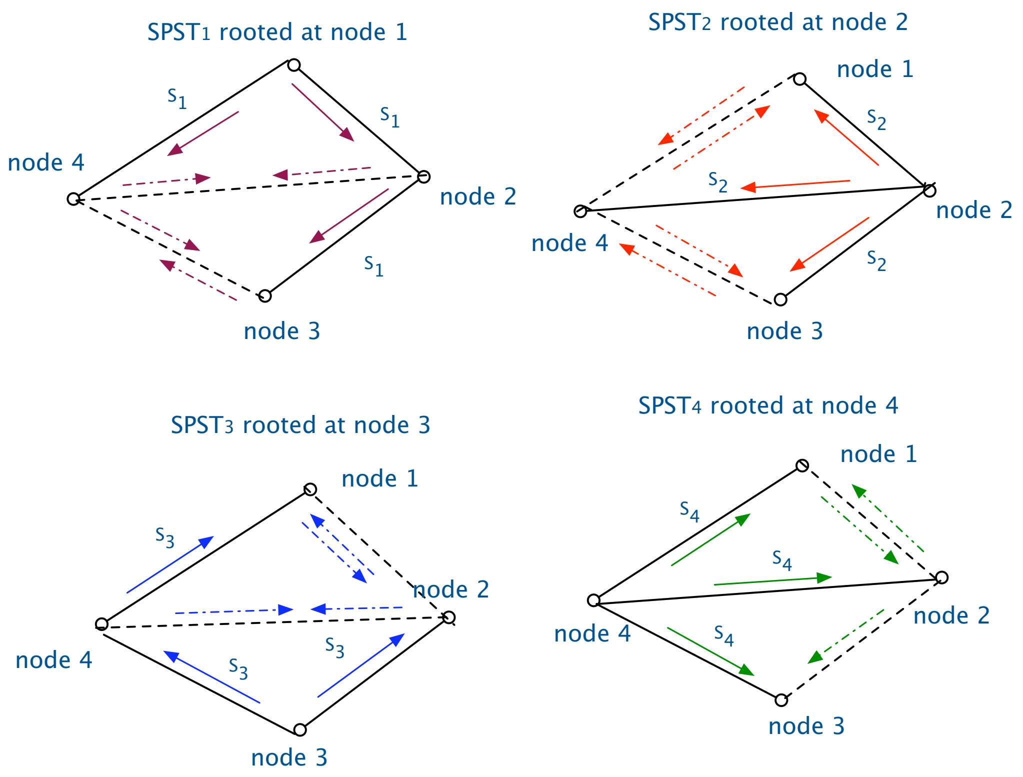


Figure 8. Packet forwarding paths by applying RPF at every node.

By employing RPF as outlined in Fig. 7, a node forwards every media segment to its neighboring nodes only once. Let us use rl_i^k to represent the number of copies of media segment s_i (originated from node i) to be sent by a node k to its neighbors. An optimal media segment vector based on RPF paths can be solved using Eq. (5) as follows.

$$\begin{aligned} \min \quad & y \qquad\qquad (5)\\ \text{s.t.} \quad & \frac{s_k}{d_k} + \sum_{\text{source node } i=1}^{n} rl_i^k \times \frac{s_i}{u_k} \le y, \\ & \quad \text{for } k = 1, 2, \cdots, n \\ & \sum_{k=1}^{n} s_k = S \\ & s_k \ge 0, \quad \text{for } k = 1, 2, \cdots, n \end{aligned}$$

where rl_i^k is calculated as

$$rl_i^k = \begin{cases} \text{number of links} & \text{if } i = k \\ \text{number of links minus 1} & \text{if } i \ne k \end{cases}$$

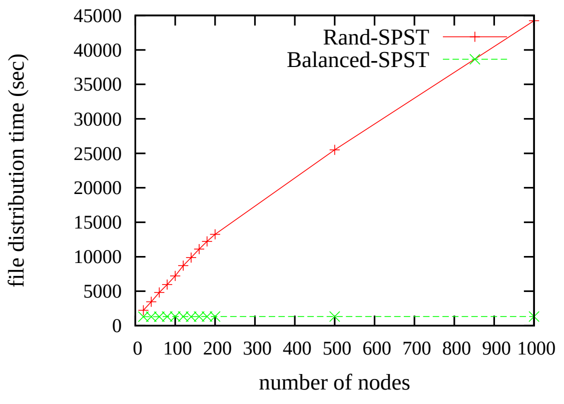
Applying Eq. (5) to the example in Fig. 8, we have the following numerical problem.

$$\begin{aligned} \min \quad & T \\ \text{s.t.} \quad & \frac{s_1}{600} + 2\frac{s_1}{300} + \frac{s_2}{300} + \frac{s_3}{300} + \frac{s_4}{300} \le T \\ & \frac{s_2}{750} + 3\frac{s_2}{400} + 2\frac{s_1}{400} + 2\frac{s_3}{400} + 2\frac{s_4}{400} \le T \\ & \frac{s_3}{500} + 2\frac{s_3}{350} + \frac{s_1}{350} + \frac{s_2}{350} + \frac{s_4}{350} \le T \\ & \frac{s_4}{800} + 3\frac{s_4}{500} + 2\frac{s_1}{500} + 2\frac{s_2}{500} + 2\frac{s_3}{500} \le T \\ & s_1 + s_2 + s_3 + s_4 = 27000 \\ & s_1, s_2, s_3, s_4 \ge 0 \end{aligned}$$

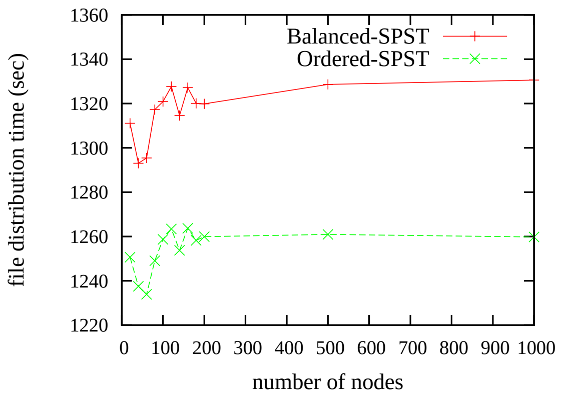
The resulted optimal media segment vector is $(s_a, s_b, s_c, s_d) = (9000, 0, 11911.8, 6088.2)$ kbits, and the optimal system distribution time is 135 seconds. It is clearly longer than 62.6 seconds needed by SPST-based distribution, when a node forwards a media segment along its spanning tree edges.

VII. Experimental Results

In this section, we evaluate the proposed optimal file distribution algorithms. In particular, we study the impact of different SPST construction algorithms on system distribution time, evaluate the optimal file segmentation algorithm, and measure the trade-off of performance and scalability between the centralized and distributed file distribution schemes. Lastly, we show how node churning rate affects system distribution times of different algorithms.

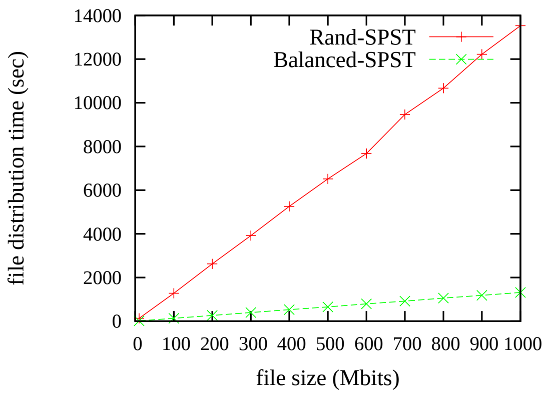


(a) Rand-SPST and Balanced-SPST

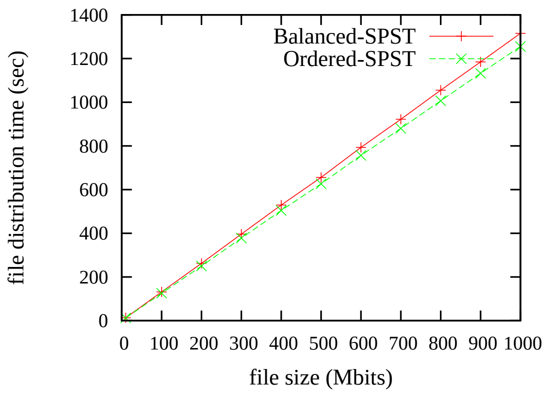


(b) Balanced-SPST and Ordered-SPST

Figure 9. System distribution time of Rand-SPST, Balanced-SPST, and Ordered-SPST, when varying the number of nodes in the system (file size = 1000 Mbits).



(a) Rand-SPST and Balanced-SPST



(b) Balanced-SPST and Ordered-SPST

Figure 10. System distribution time of Rand-SPST, Balanced-SPST, and Ordered-SPST, when varying file size (n=200).

A. Simulation Setup

We implemented our own simulation program using C++. First, we randomly generate the topology of a mesh network, including how nodes are connected, each node's uploading and downloading bandwidths. Next, we use different heuristic SPST construction algorithms to decompose the mesh network into multiple spanning trees. Given the SPSTs, we construct the linear programming problems defined in Eq. (3), Eq. (4), and Eq. (5), using the library functions provided by *lp_solve*. Last, we call *lp_solve* to find the optimal solutions to these problems. Each data point reported in the below is an average of more than 50 simulation runs.

In our experiments, we evaluate different algorithms in two ways. One type of experiments fixes the distribution file size and varies the number of nodes in the system, and the other fixes the number of nodes and varies the distribution file size. We have observed that typical BitTorrent networks in the Internet usually consist of tens to hundreds peer nodes. Therefore, we performed our experiments for node sizes from 20 to 200, with an increment of 20 nodes, in experiments that vary the number of nodes. To see how the algorithms scale, we also performed the experiments for larger systems, consisting of 500 peers and 1000 peers, respectively. When evaluating system distribution times with respect to file sizes, we choose a typical network size of 200 nodes.

B. Comparing SPST construction algorithms

We have compared the three SPST construction algorithms by varying the size of distribution files and the number of nodes in a system. Fig. 9 compares system distribution times of Rand-SPST, Balanced-SPST, and Ordered-SPST, with different number of nodes in the system. The size of the file to be distributed is 1000 Mbits. Fig. 9a shows that Balanced-SPST consistently takes shorter time than Rand-SPST. Its distribution time is almost constant, regardless of the number of nodes in the system. On the other hand, the file distribution time of Rand-SPST increases linearly with the number of nodes. As a result, the difference in system distribution time between Balanced-SPST and Rand-SPST significantly increases as the number of nodes increases. The reason is as follows. When a system becomes larger and larger, the file transmission load resulted from Rand-SPST construction becomes more unevenly distributed to nodes. In contrast, the transmission load by Balanced-SPST algorithm is approximately the same, because Balanced-SPST monitors and balances the load distributed to each node. Fig. 9b shows the impact of initial node orderings of SPST construction on the file distribution time. By initially

TABLE I.
COMPARISON OF OPTIMAL FILE SEGMENTATION AND EQUAL FILE DIVISION, WHEN VARYING NUMBER OF NODES, TO DISTRIBUTE A FILE OF 1000 MBITS.

n	**Rand-SPST**		**Balanced-SPST**		**Ordered-SPST**	
	optimal	**equal**	**optimal**	**equal**	**optimal**	**equal**
20	2261.00	5906.40	1292.94	3864.00	1243.15	3061.92
40	3633.44	10409.47	1315.05	4371.54	1246.36	3187.99
60	4901.42	14661.56	1301.05	4526.82	1245.39	3189.07
80	6133.66	19027.18	1311.48	4665.80	1252.54	3247.69
100	7329.03	24661.79	1309.29	4845.38	1249.23	3291.12
120	8758.12	32052.46	1315.56	5081.92	1256.21	3342.08
140	9308.69	32726.02	1301.89	5045.95	1246.03	3383.71
160	10660.61	39868.16	1314.67	5196.82	1255.92	3394.06
180	12570.84	50780.51	1318.85	5250.28	1259.94	3427.84
200	13485.73	54130.83	1309.99	5336.70	1250.13	3427.49
500	25513.52	134160.00	1298.61	5602.61	1243.92	3390.72
1000	44227.03	215122.00	1269.55	5633.55	1228.79	3469.42

TABLE II.
COMPARISON OF OPTIMAL FILE SEGMENTATION AND EQUAL FILE DIVISION, WHEN VARYING THE SIZE OF FILE, IN A SYSTEM OF 200 NODES.

S (Mbits)	**Rand-SPST**		**Balanced-SPST**		**Ordered-SPST**	
	optimal	**equal**	**optimal**	**equal**	**optimal**	**equal**
10	129.73	491.23	13.11	52.64	12.53	34.41
100	1292.93	5237.78	131.49	527.89	125.47	345.87
200	2686.47	9864.05	263.39	1080.03	251.35	689.60
300	3876.99	15138.31	397.72	1576.51	379.23	1034.62
400	5351.63	18881.27	528.45	2109.63	504.06	1377.23
500	6454.40	24408.97	661.63	2633.99	632.05	1737.29
600	7537.73	27556.82	788.85	3142.40	753.74	2059.98
700	9343.12	35061.79	921.38	3671.59	880.04	2430.72
800	10989.09	42211.90	1046.93	4281.31	998.96	2787.88
900	11727.25	44831.47	1184.13	4775.24	1130.51	3130.84
1000	12774.70	43850.75	1310.90	5297.73	1251.91	3431.24

ordering the nodes in an increasing order of its uploading capacity, the system distribution time decreases by 4.5% compared to Balanced-SPST without node ordering.

Fig. 10 compares the system distribution times of Rand-SPST, Balanced-SPST, and Ordered-SPST, when files of different sizes are distributed. In the experiments, there are 200 nodes in the P2P system. We see similar results as in Fig. 9. Both Balanced-SPST and Ordered-SPST significantly outperform Rand-SPST, as the distributed file size increases. Moreover, Ordered-SPST consistently takes approximately 5% less time than Balanced-SPST.

Therefore, among the three SPST construction algorithms, Ordered-SPST achieves the best system distribution time.

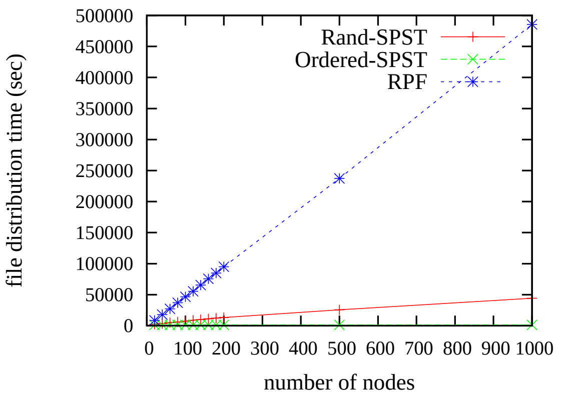
C. Optimal File Segmentation

We evaluate the optimal file segmentation algorithm discussed in Section IV-B by comparing it with a simple scheme that divides a file into equal-sized segments. This simple scheme is commonly used in practical BitTorrent-like overlay systems. Table I gives a side-by-side comparison of the two schemes when three SPST construction algorithms are used. We observe that with increasingly balanced load in the topologies based on Rand-SPST, Balanced-SPST, and Ordered-SPST, both equal file division scheme and the optimal file segmentation algorithm achieve smaller system distribution time. We can observe similar results in Table II, which compares the two file segmentation schemes, when varying the size of file to be distributed, in a system of 200 nodes.

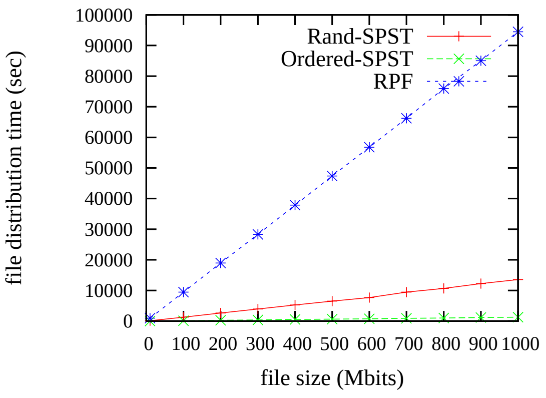
In all the experiments listed in two tables, optimal file segmentation algorithm consistently outperforms equal file division scheme. The optimal file segmentation algorithm only takes 25-35% of the time used by equal file division scheme. The optimal file segmentation solution is obtained from a linear programming formulation based on Eqn. (3). It is solved very efficiently by *lp_solve*, taking only a fraction of a second. Hence the performance gain is achieved with negligible computational overhead.

D. Reverse Path Forwarding

The three SPST based algorithms require every node in a system to keep track of its forwarding paths in all the SPSTs. To reduce the amount of state information kept in every node, reverse path forwarding can be used independently by each node to determine whether to forward a segment, and if so, the set of nodes to forward the segment. To evaluate RPF's performance, we compared the system distribution time of Rand-SPST, Ordered-SPST, and SPF, when varying the number of nodes in the system (see Fig. 11a) and varying the size of distributed files (see Fig. 11b). As the size and distribution load in the system increase, RPF takes about 7-8 times longer than Rand-SPST, and 75 times longer than Ordered-SPST to complete a file distribution. This suggests that BitTorrent-like systems, in which file pieces can be duplicated among

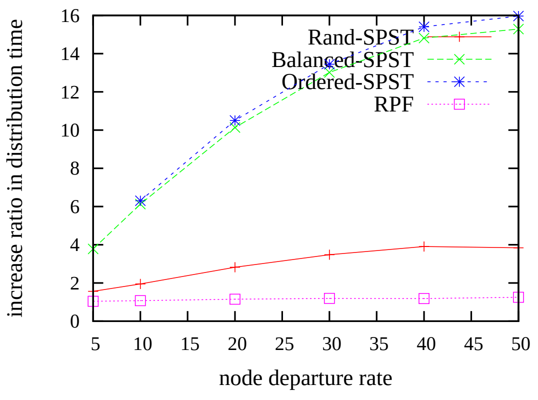


(a) varying number of nodes

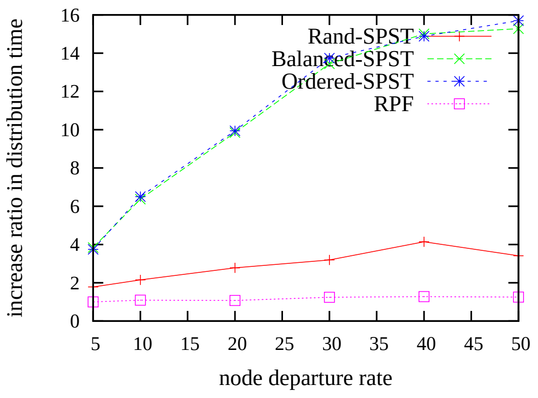


(b) varying file size

Figure 11. System distribution time of Rand-SPST, Ordered-SPST, and RPF, when (a) varying number of nodes in the system (file size = 1000 Mbits) and (b) varying file size (n=200).



(a) file size = 100 Mbits



(b) file size = 1000 Mbits

Figure 12. Increase ratio of Rand-SPST, Balanced-SPST, Ordered-SPST, and RPF when varying peer churning rates: (a) file size = 100 Mbits, and (b) file size = 1000 Mbits.

peers, take much longer system distribution time than the file distribution model without piece redundancy.

The comparisons present an interesting trade-off between the performance of system distribution and scalability in implementation. Adopting a controlled flooding approach like RPF makes an overlay system easy to deploy and scale. On the other hand, using this scalable algorithm compromises system performance. The selection of an algorithm largely depends on system setup, application requirements, and deployment considerations.

E. Impact of Node Churning

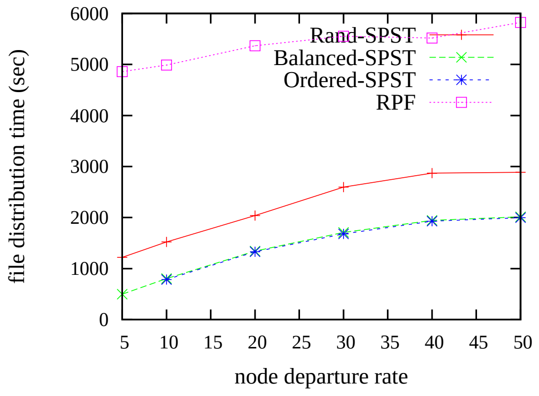
In this section, we study how different distribution methods react to peer churning. We model peer departure as a random process and evaluate how the distribution algorithms perform with different departure rates.

In the experiments, given a departure rate, nodes are chosen at random to leave the system at random intervals. In Fig. 12, we plot the increase ratios of distribution times of four algorithms: Rand-SPST, Balanced-SPST, Ordered-SPST, and RPF. We have done extensive comparisons using different distribution file sizes, ranging from 10 Mbits to 1000 Mbits. Due to space limits, we only include the comparisons of two representative cases here: a small file size of 100 Mbits in Fig. 12a and a large file size of 1000 Mbits in Fig. 12b. The increase ratio of each algorithm is calculated as follows:

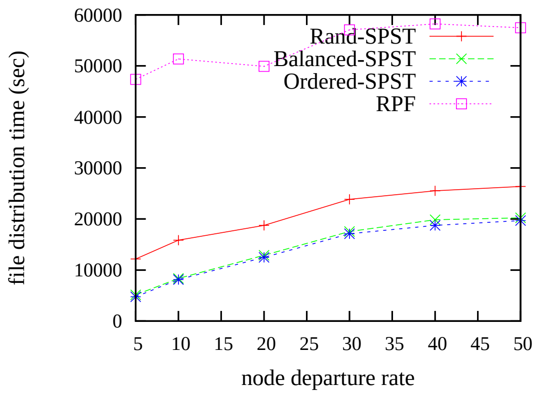
$$\text{increase ratio} = \frac{\text{distribution time with node departure}}{\text{distribution time without node departure}} \tag{6}$$

Fig. 12 shows that RPF produces the smallest increase ratio among the four algorithms, benefiting from the redundancy of packets in distribution networks. In three SPST-based distributions, all of the unfinished file segments due to node departures need to be re-assigned and re-distributed in the next cycle. In contrast, in RPF-based distribution, some of these unfinished segments could have propagated through alternative routes to some remaining nodes through its controlled flooding. They don't need to be resent in the next cycle.

Rand-SPST has the second smallest increase ratio. As seen from Eq. (6), the increase ratio is a function of two parameters: the increase of distribution time due to node departure and the baseline distribution time without node departure. The increase of distribution time consists of two components: (1) R-time – the time to redistribute the segments that are assigned to and delivered by these departure nodes as the root nodes of SPSTs, and (2) I-time – the time to redistribute the segments to be forwarded by these departure nodes as the internal nodes of SPSTs.



(a) file size = 100 Mbits



(b) file size = 1000 Mbits

Figure 13. File distribution time of Rand-SPST, Balanced-SPST, Ordered-SPST, and RPF, when varying peer churning rates: (a) file size = 100 Mbits, and (b) file size = 1000 Mbits.

The segments in (1) have not been forwarded to any other nodes in the network, and thus should be requested from an original server. The redistribution process of (1) can be considered as a distribution of a "new" file, so the comparison of R-time should follow similar trends as shown in Fig. 9 and Fig. 10. R-time taken by Ordered-SPST is the smallest, followed by Balanced-SPST and then Rand-SPST. On the other hand, the segments in (2) are available in other nodes of the network. For each of these partially forwarded segments, we independently construct a supplying set and apply Dijkstras algorithm to build forwarding paths. Therefore, I-time is comparable for the three SPST algorithms. Since the increase of distribution time is dominated by I-time, the total increase of distribution time is almost the same for the three SPST algorithms. Because Rand-SPST has a larger baseline distribution time without node departure, thus it gives a smaller increase ratio, compared to Balanced-SPST and Ordered-SPST.

Fig. 13 compares the file distribution times of the four algorithms for different node departure rates. Despite larger increase ratios, Balanced-SPST and Ordered-SPST still have the lowest distribution times among the algorithms, taking about 50% to 66% of the time used by Rand-SPST and 10% to 33% of the time used by RPF.

F. *Findings*

The distribution times achieved by two balanced SPST algorithms, Balanced-SPST and Ordered-SPST, provide theoretical baselines on the minimum time needed by practical overlay systems to distribute a file. Such optimized distribution times are obtained based on three design choices. First, each node is assigned a unique file segment for distribution, so there is no packet redundancy in a distribution network. Second, each node fully contributes its uploading bandwidth using a "push-based" model and does not leave the network before completing its forwarding tasks. And third, a file is optimally divided into unequal-sized segments by solving a constrained optimization problem. Practical overlay systems often deviate from the above design choices, due to the considerations in implementation complexity, scalability, and resilience to network dynamics. For example, BitTorrent systems allow packet redundancy in a network with multiple nodes possessing the same file segment. Many CDNs and P2P systems simply divide a file into equal-sized segments for distribution. The experiments help us gain insights on the impact of each of the three design choices.

- Reverse path forwarding uses controlled flooding to build file forwarding paths. Each node simply forwards packets to its neighbors in a loop-free fashion. This algorithm is easy to implement and more scalable compared to SPST-based distributions. However, a packet could be sent unnecessarily to other nodes that have already received the packet. In the experiments, we find that the system distribution time increases up to 75 times compared to the two optimized SPST algorithms. In practice, we want to strike a balance in the file distribution performance and system scalability.
- The optimal way to divide a file into unequal-sized segments can be found by efficiently solving a linear programming problem. Compared to the usual way of dividing a file into equal-sized segments, this optimization results in a performance gain of 3-4 times, without incurring much computational overhead.
- When considering the scenarios, where nodes dynamically leave a distribution system, RPF is shown to be the most resilient one, measured by the increase ratio of distribution time. However, the overall distribution time needed by RPF is still much larger than the three SPST algorithms.

VIII. CONCLUSIONS

Our theoretical analysis and experiment evaluation in this paper complement related work: we study the performance of more practical mesh-based topology in P2P and CDN networks. In such networks, nodes are partially connected and organized in a mesh topology. By removing piece redundancy among nodes, leveraging the largest contribution of each node's uploading bandwidth, and optimizing file segmentations, we obtain a lower bound on the system distribution time of a mesh-based overlay network. We find that the design of an overlay network presents interesting trade-offs among efficiency, resilience, and scalability. SPST-based distribution algorithms result in efficient system distribution times, whereas SPF algorithm is scalable and resilient to dynamic node departures. In the future, we will develop a more efficient algorithm than RPF to scale in large systems and will implement the optimal file segmentation algorithm in practical P2P and CDN systems and evaluate them on Internet testbeds such as PlanetLab.

Acknowledgment

This work was supported in part by NSF CAREER (CNS-0546870) and NSF TRUST (CCF - 0424422) – Team for Research in Ubiquitous Secure Technology.

References

[1] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *Communications Surveys & Tutorials, IEEE*, vol. 7, no. 2, pp. 72–93, Quarter 2005.

[2] Bittorrent. [Online]. Available: http://www.bittorrent.com/

[3] B. Cohen, "Incentives build robustness in BitTorrent," in *Proc. First Workshop on Economics of Peer-to-Peer Systems*, May 2003, pp. 251–260.

[4] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent network's performance mechanisms," in *Proc. IEEE INFOCOM*, Barcelona, Spain, April 2006, pp. 1–12.

[5] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 314–329, 2003.

[6] M. Izal, G. Urvoy-keller, E. W. Biersack, P. A. Felber, and A. A. Hamra, "Dissecting bittorrent: Five months in a torrent's lifetime," in *Proc. Fifth Passive and Active Measurements Workshop (PAM '04)*, April 2004, pp. 1–11.

[7] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proc. Multimedia Computing and Networking*, 2002. [Online]. Available: http://citeseer.ist.psu.edu/saroiu02measurement.html

[8] M. Feldman, K. Lai, I. Stoica, and J. Chuang, "Robust incentive techniques for peer-to-peer networks," in *Proc. the 5th ACM conference on Electronic commerce*, New York, NY, USA, 2004, pp. 102–111.

[9] P. Golle, K. Leyton-Brown, I. Mironov, and M. Lillibridge, "Incentives for sharing in peer-to-peer networks," in *Proc. the Second International Workshop on Electronic Commerce*, London, UK, 2001, pp. 75–87.

[10] D. Levin, R. Sherwood, and B. Bhattacharjee, "Fair file swarming with fox," in *Proc. Int'l Workshop on Peer-to-Peer Systems (IPTPS)*, 2006.

[11] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster, "To share or not to share: an analysis of incentives to contribute in collaborative file sharing environments," in *Proc. Workshop on Economics of Peer-to-Peer Systems*, 2003.

[12] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "A performance study of bittorrent-like peer-to-peer systems," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 155 – 169, 2007. [Online]. Available: http://dx.doi.org/10.1109/JSAC.2007.070116

[13] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proc. ACM SIGCOMM*. New York, NY, USA: ACM, 2004, pp. 367–378.

[14] K. Ramachandran and B. Sikdar, "An analytic framework for modeling peer to peer networks," in *Proc. IEEE INFOCOM*, March 2005, pp. 2159–2169.

[15] L. Cherkasova and J. Lee, "Fastreplica: Efficient large file distribution within content delivery networks," in *Proc. 4th USENIX Symposium on Internet Technologies and Systems*, Seattle, Washington, March 2003.

[16] L. Cherkasova, "Optimizing the reliable distribution of large files within cdns," in *Proc. IEEE Symposium on Computers and Communications*, June 2005, pp. 692–697.

[17] R. Kumar and K. W. Ross, "Peer-assisted file distribution: The minimum distribution time," in *Proc. 1st IEEE Workshop on Hot Topics in Web Systems and Technologies*, Nov 2006, pp. 1–11.

[18] J. Mundinger, R. Weber, and G. Weiss, "Analysis of peer-to-peer file dissemination," *SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 3, pp. 12–14, 2006.

[19] J. S. K. Chan, V. O. K. Li, and K.-S. Lui, "Performance comparison of scheduling algorithms for peer-to-peer collaborative file distribution," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 146 – 154, 2007. [Online]. Available: http://dx.doi.org/10.1109/JSAC.2007.070115

[20] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1979.

[21] "lp_solve," http://sourceforge.net/projects/lpsolve.

[22] Y. K. Dalal and R. M. Metcalfe, "Reverse path forwarding of broadcast packets," *Communications of the ACM*, vol. 21, no. 12, pp. 1040–1048, 1978.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 2009, Ed. The MIT Press, 2009.