Fast Replication in Content Distribution Overlays

Samrat Ganguly, Akhilesh Saxena, Sudeept Bhatnagar, Suman Banerjee, Rauf Izmailov

***Abstract*— We present SPIDER – a system for fast replication or distribution of large content from a single source to multiple sites interconnected over Internet or via a private network. In order to exploit spatial diversity of the underlying network, SPIDER uses an overlay structure composed of dedicated Transit Nodes (TNs). The data transport mechanism in SPIDER leverages this overlay structure to provide a coordinated approach that minimizes the maximum time to replicate to all destination sites (the makespan of content replication). In order to achieve this objective, SPIDER employs two orthogonal components: a) creation of multiple dynamic distribution trees using the transit nodes b) end-to-end reliable data transport with flow control on these trees by chaining point-to-point TCPs. We further present simulations based results to quantify benefits of tree construction algorithms in random topologies. We evaluate the real implementation of the SPIDER in PlanetLab and observe a 2-6 times speed up compared to different existing schemes.**

***Index Terms*— System Design, Mathematical programming/optimization, Graph Theory, Experimentation with real networks/Testbeds.**

I. Introduction

The recent emergence of new applications in the entertainment, business and scientific communities has led to a tremendous growth in the size of data sets in the recent past [1] and has necessitated the research in the area of fast bulk data transfer and replication. For example, in the entertainment community, content providers are now interested in transferring very large content such as digital video programming or video-on-demand movies over the Internet from a central server to geographically distributed edge servers for providing on demand streaming to their clients. Similarly, movie distributors are considering the use of the Internet to transport movies from their central location to movie theaters around the country and the world. Analogously, wide-area storage backup systems are interested in transporting large volumes of data to be archived from a single location to multiple remote data centers over the Internet. Large software companies have to periodically update their various mirror sites with the latest releases of their software products. In e-science collaboration projects, large experimental data sets need to be transferred to geographically distributed locations for analysis. For all of these applications, the key performance metric is the total time needed to replicate large data sets. This paper presents practical approaches to extract more bandwidth from underlying network in designing a system that can be used to accelerate the data replication.

S. Ganguly, A. Saxena, S. Bhatnagar, and R. Izmailov are with the Broadband and Mobile Networking Department,NEC Laboratories, Princeton, NJ, 08540, USA. Emails: {samrat,saxena,sudeept,rauf}@nec-labs.com. S. Banerjee is with the Department of Computer Sciences, University of Wisconsin, Madison, WI 53706, USA. Email: suman@cs.wisc.edu

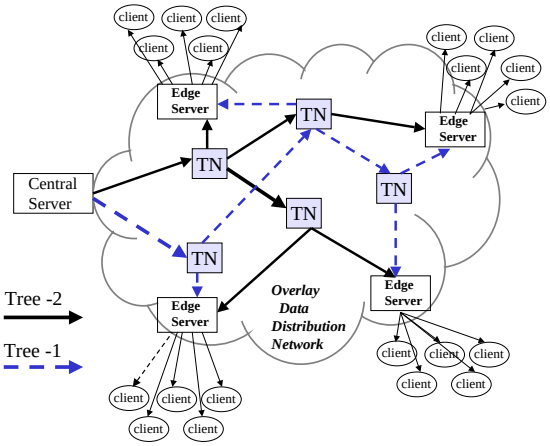

Fig. 1. Content distribution overlay using transit nodes; example scenario: source replicates content at edge servers using multiple trees – Tree-1 (solid) and Tree-2 (dashed)

Our approach is based on an overlay architecture composed of a number of dedicated transit nodes (TNs) as shown in Figure 1. The source in these applications are content publishers, data repositories, or central servers. In this architecture, the content is not directly pushed to the clients. Instead the source replicates the large volume of data to a relatively small set of edge servers, caches, or data centers. End-users, that are the actual clients of this data, subsequently downloads it from these edge servers. A similar architecture is employed by popular CDNs, like Akamai, to distribute large (more than 100 MB) movies (bmwfilms [2]) to edge servers. The edge servers subsequently stream these movies to millions of clients. With the steady increase in the size of the data sets being distributed across such CDNs from a single source to multiple edge servers (destinations), the transfer latency

has gained importance. Typically, the source of the data initiates this *replication process* of a large volume of data to a known set of destinations at a specific time. The objective then becomes to minimize the makespan, i.e., to minimize the total time to replicate content from the source to these chosen destination set. To achieve such data replication our proposed architecture uses a set of special intermediaries called Transit Nodes (TNs), that can be deployed at opportunistic locations in the network, e.g., GigaPoPs, to improve performance.

We distinguish such a data replication application from the file sharing applications that are popular in the peer-to-peer (P2P) domain. P2P file sharing applications, (like Kazaa, Emule) are receiver-driven, i.e., an individual receiver initiates data download asynchronously, independently of all other requests of the same content. Most of these schemes allow individual clients to download data opportunistically from multiple sites towards minimizing download time. To facilitate such a download Kazaa organizes the clients into an overlay P2P network. The overlay changes with time as nodes dynamically join and leave the system.

The following are key differences between our proposed work and prior work in the context of P2P downloads: (1) We are examining a different problem — *how do we efficiently replicate large content from a single source to pre-determined multiple destinations in a coordinated and synchronous fashion.* In this sender-driven scenario, it is likely that the sender, the Transit Nodes and all the destinations will implement a coordinated solution that optimizes the makespan. This is very unlike the receiver-driven problem in which protocols can be designed for an individual client to maximize its own data rate by downloading content from the source and other peers that are available in the system, without necessarily trying to minimize the global objective function. (2) In P2P download scenarios there are typically a large number of peers many of which are located across low bandwidth access links. Hence in most proposed P2P download systems, a client randomly discovers *a few* other peers through simple randomized techniques. Subsequently due to bandwidth limitations they perform a coarse-grained measurement to choose a subset of these peers for parallel download. In our fast replication scenario, the source, Transit Nodes, and destinations are part of a high-bandwidth CDN. The total number of such nodes is much lower than the number of peers in P2P systems. Therefore in our case we are able to probe the suitability of all possible paths between these nodes and perform more accurate measurements on them. Note that a CDN (e.g., Akamai) already have such measurement infrastructure in place [30]. In particular we model this problem as an optimization problem (which is NP-Hard) and hence propose heuristics that perform well with respect to upper-bounds on the optimal. (3) In our replication case we consider data distribution in very large volumes (GBs or TBs). Hence we can leverage this long-lived nature of this replication process in our design.

A. Content distribution overlay

We consider a CDN architecture as shown in Figure 1 where a set of intermediaries act as TNs. The TNs are placed at opportunistic locations in the network that organize themselves into a content distribution overlay and are used to replicate and forward data. These nodes typically are servers with fast cache and high access bandwidth. The advantages of TNs are two-fold: (1) They allow us to better exploit available spatial diversity by using multiple alternative paths available within the network from the source to the destinations and thus reduce data transfer time, (2) They allow us to better exploit alternate paths that can be better than the direct unicast paths. Recent research (Detour [13], RON [14]) has shown that in many cases alternate overlay paths have better latency and throughput characteristics than direct IP paths. For example in specific experiments the authors in [13] have shown that in 30% of direct IP paths have better alternate paths with respect to the round trip time metric and 70% of direct IP paths have better alternate paths with respect to the loss metric. RON uses such alternate paths for fault tolerance.

A common approach to distribute data using an overlay is to organize the set of nodes into an overlay tree rooted at the source. Various optimized multicast tree creation algorithms have also been proposed in [15], [16], [17], [10]. However, as pointed out in [4], a single multicast tree based data forwarding cannot achieve high throughput as bandwidth monotonically decrease with depth of the tree. Additionally, decreasing tree depth by increasing tree fan-out is not necessarily a good option as it can increase the forwarding load on the transit nodes, as noted in the SplitStream work [6].

B. Contributions

We present a stand-alone content distribution system, SPIDER, based on **SP**atial **I**ndirection for path **D**iversity for **E**xpedited **R**eplication. SPIDER utilizes the transit nodes in creating multiple multicast trees and coordinates the transport of data on these trees to the given set of destinations. In SPIDER, the original data is stripped into equal sized blocks and reliably transported to destination using the multicast distribution trees and aggregated into

the original file at each destination. We present the design of SPIDER which has two core components: a) computation of multiple distribution trees; b) flow control in data distribution among trees.

In Section II we formulate the data replication problem as an optimization problem — minimizing the replication makespan. Since this problem is NP-Hard, we propose a polynomial-time approximation (Appendix). Although this approximation is polynomial-time, it is still computationally expensive and hence we define a heuristic algorithm (called MBST) for computing multiple trees based on bandwidth measurements. The tree computation method tries to maximize the total throughput achieved using all trees. We compare the proposed algorithm with other alternate algorithms as well as an upper bound of the optimal.

We subsequently present the data distribution architecture of SPIDER that defines flow control mechanisms at block level and gracefully adapts to variations of bottleneck bandwidth of a given tree. At the same time, the flow control mechanism ensures an efficient load balancing among the trees based on the long-term average bottleneck bandwidth of the tree.

Some of the salient features of the SPIDER system are: a) trees can be reconfigured quickly to changing network conditions without losses, b) it is resilient to Transit Node failures, c) tree creation can be based on chaining of any point-to-point transport protocol (we use TCP currently) d) sharing a *single* TCP connection between edges in different trees passing through same pair of nodes, which leads to TCP friendliness in the network. The system is easy to use (akinto simple FTP) and needs installation of the same SPIDER servers at all the participating nodes (destination and TNs). Finally, we have made the tool available at [18] for community use.

We have evaluated the performance of the real implementation of SPIDER on the PlanetLab (see http://www.planet-lab.org). In our experiments we have examined the performance for transfers of different sizes of data on different topologies on the PlanetLab with diverse sets of nodes (diversity of geographical locations and their access bandwidths). In these experiments we compared the performance of SPIDER to different schemes, including multiple unicast TCP sessions in parallel and existing fast file download tools such as BitTorrent [8] and Slurpie [9]. Our results indicates that SPIDER can achieve speed-up of up to five-six times compared to multi-unicast approaches. SPIDER provides two times speedup compared to Slurpie and around four times speed up compared to BitTorrent.

C. Related Work

In the recent past, a significant amount of research has been directed towards developing protocols and mechanisms for fast bulk data delivery. A number of such research efforts were focussed on unicast data transfers, e.g., Fast-TCP [19], XCP [20], gridFTP [21], blast-UDP [22], Digital Fountain [23]. SPIDER can use any of these schemes as the underlying transport protocol between each pair of nodes on the overlay.

At the same time, with the tremendous growth of peer-to-peer networks for file sharing, various protocols were developed to minimize the file download time. These protocols are based on downloading a file in parallel from more than one peer. Simple versions of such approach already exist in current Kazaa and E-mule P2P networks.

Various improved solutions were proposed in the context of P2P network, such as BitTorrent [8], Bullet [7], informed content delivery [4], Slurpie [9], ROMA [10], SplitStream [6]. In most of these approaches, it is the client which is responsible for selecting a set of others of peers from where they downloads a complementary set of blocks forming the original file. In different protocols, such a set of peers are chosen either randomly or based on selective bandwidth estimation for a few peers. Hence these solutions are particularly tailored for P2P download applications, where clients join and leave in ad-hoc fashion. In contrast our work focusses on coordinated replication in Content Distribution Networks in which the objective of the source, Transit Nodes, and destinations is to minimize the makespan of the replication process. We demonstrate the difference of these techniques and ours by experimental comparisons to two of these schemes (BitTorrent, which is widely used today, and Slurpie). Some of these approaches, e.g., ROMA, use a single overlay multicast tree for data distribution and hence cannot exploit path diversity in the network. SplitStream, on the other hand, proposes the use of multiple trees for data transfer with the goal of balancing the forwarding load at intermediate peers (Transit Node in our case). Their solution is particularly geared towards scenarios where the uplink and downlink *access* bandwidth of the peers is the bottleneck. For high-bandwidth CDN architectures, this is not necessarily the case because the destinations are edge servers that are placed in locations with high access bandwidth. Additionally empirical studies in [11] show, end-to-end path bottlenecks frequently occur in intra-AS and inter-AS peering links, and not just at end-point access links.

Fast Replica [24] is another solution which essentially constructs multiple (K) trees. The file is split into K

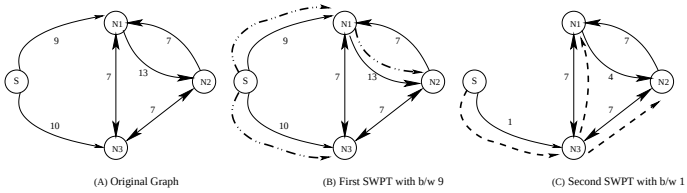

Fig. 2. An example showing the failure of shortest widest path algorithm in finding the best trees.

stripes and each stripe is sent along one of the K trees. However, their tree construction process (unlike ours) does not take into account bandwidth of different paths into account and hence their approach may lead to low quality trees. Additionally the amount of data transported on each of these trees does not depend on the tree bottleneck. Consequently Fast Replica would be very inefficient in a scenario where two trees are used and one tree has ten times the bandwidth of the other.

Apart from bulk data delivery, the use of path diversity also finds applications in the context of real-time streaming. For these applications, various coding solutions were proposed such as [12] with the main purpose of increasing decoding quality and not minimizing transfer time. Path diversity was also proposed in [3], where a client establishes parallel connections to multiple sites to download a file. However, our objective is just the reverse - how to replicate from one source to multiple destinations.

D. Organization of the paper

In the next two sections we present the two important components of SPIDER, namely creation of multiple trees and the data distribution mechanism using these trees. We present the results to evaluate the performance of SPIDER based on simulation and experimentation in PlanetLab. Finally, we present conclusions and future directions of work.

II. Tree computation algorithm based on bandwidth estimation

SPIDER has two orthogonal components: (1) tree computation, and (2) reliable data transport on the computed trees. Such a split allows us to efficiently deploy this system over the Internet as well as over private networks. In many private networks, the topology along with the link bandwidth maybe known a-priori and therefore the distribution trees can be pre-computed and can be passed to SPIDER as parameters. On the other hand, in Internet, the trees can be created dynamically based on current bandwidth availability and can be reconfigured in the middle of replication process. In this section, we present the algorithm used to compute multiple distribution trees in SPIDER. The distribution system of SPIDER (described in the next section) takes these trees as input parameters and performs data replication using them.

A. Formal Problem

The goal in SPIDER is to minimize the *makespan* in data replication, where makespan is defined as the total amount of time between the moment the data transfer is initiated and the moment when the data download completes at the last destination.

In this section, we focus on algorithms for constructing optimized multiple multicast trees on a capacity constrained graph. Let s denote the source node and D denote the set of destination nodes in a graph $G = (N, A)$, where N is the set of vertices and A is the set of edges. Given a capacity constrained graph $G = (V, E)$, a source s and a set of destinations $D \in V - \{s\}$, find

- a set $\mathcal{A}$ of directed rooted trees $\{T_1, \ldots, T_n\}$ from the source s spanning the destinations D
- a bandwidth assignment $\beta = \{\beta(T_1), \ldots, \beta(T_n)\}$ such that the total bandwidth $B_{max} = \beta(T_1) + \ldots + \beta(T_n)$ is maximized and the capacity constraints are met.

The special case of $|D| = 1$ can be solved by the max-flow-min-cut theorem.

In the graph G, let $F(d)$ denote the max-flow value from the source node s to a destination node d, where $d \in N - \{s\}$. Let $\lambda(G) = \min_{d \in N - \{s\}} F(d)$ denote the minimum of the max-flow values from the source s to all the destinations $N - \{s\}$. If $D = N - \{s\}$, the total bandwidth B from any set of trees cannot exceed $\lambda(G)$. Further refinement of this observation comes from Edmond's packing theorem [27], which states that in an integral capacity scenario, the maximum value of B is $B_{max} = \lambda(G)$. The formulated problem can be considered as *maximum packing* of directed rooted spanning trees in a graph. There exists a polynomial time algorithm providing optimal solution to this problem based on [27] as can be found in [28]. However, the problem in the general non-spanning case, where $D \subset N - \{s\}$), is NP-Hard. The solution for this scenario reduces to solving a set of Steiner tree problem and thereby has same approximation bound as Steiner case. We present a sketch of the approximation algorithm for the general case in appendix based on results in [29].

As we may require fast computation of trees in order to adapt to changing bottleneck bandwidth, we need a practical and efficient algorithm with reduced

complexity, than what the approximation algorithm in the Appendix provides. Therefore, we describe a heuristic algorithm to compute the multiple trees, given the estimated bandwidth between different pairs of nodes. In the next two subsections we show the limitations of simple tree computation algorithms and then describe how our algorithm overcomes these shortcomings.

Let the bandwidth between nodes i and j is given by b_{ij} (and is the weight of the arc between i and j in G). If the bandwidth between nodes i and j is not known, we set b_{ij} to 0. Along with the bandwidth, the outgoing access bandwidth E_i of each node i is assumed to be known. If E_i is not known, then we set E_i equal to the sum of outgoing bandwidths from i to all other nodes ($E_i = \sum_j b_{ij}$ where $j \in N$). In all cases, $E_i \geq b_{ij}$ (for all $j \in N$) since the outgoing bandwidth to an individual node cannot be more than the access bandwidth of the node.

B. A Naive Approach - Shortest Widest Path Tree (SWPT)

Using the above information, one could conceive several algorithms to construct multiple source-rooted trees. We now look at a naive tree constructing algorithm and use it to identify the issues that need to be accounted for in designing a good tree construction algorithm. Note that the tree construction algorithms have to create a directed tree rooted at source, however, for the purpose of exposition we refer to this as a tree.

Since the amount of bandwidth that a tree could provide for transfer is limited by its bottleneck bandwidth, a natural candidate for tree construction is the Shortest Widest Path Tree (SWPT) algorithm (which is a variant of the shortest path algorithm that maximizes the bottleneck bandwidth from source to any destination). A simple algorithm to create multiple trees is obtained by repeated application of the SWPT algorithm.

- Find a source-rooted SWPT in the graph.
- If an SWPT is found, remove the edges in the SWPT from the graph and repeat first step.
- Otherwise report all the SWPTs found and exit.

While this algorithm is simple, the key problem with this greedy algorithm is that in each iteration it only considers the present tree. There is no notion of leaving enough bandwidth for the trees that would be created in subsequent iterations. This is best illustrated using the example shown in Figure 2. The example has a source S and three destination nodes labelled N_1, N_2, N_3. The numbers shown in the figure are the bandwidths of the edges. Figure 2(B) shows the SWPT on the original graph. The bottleneck capacity of this tree is 9 units.

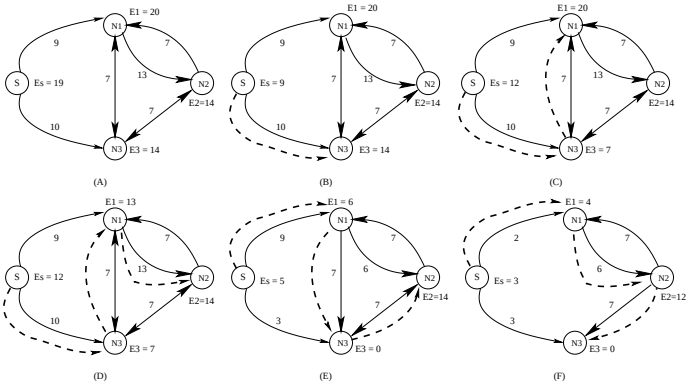

Fig. 3. An example showing the use of our algorithm in creating multiple trees.

We eliminate the 9 units utilized by this tree to get the network shown in figure 2(C). Notice that the execution of SWPT algorithm uses both of the outgoing links from the source thus leaving only 1 unit of outgoing bandwidth from the source for future. Thus, the next tree created has only 1 unit of bandwidth as shown in figure 2(C). It is important to note that the minimum incoming bandwidth into the destination nodes is at least 16 units and the repeated use of shortest widest path results in a use of 10 units of bandwidth through the two trees.

This example also show that the use of a maximum bandwidth spanning tree also results in bandwidth underutilization. It can be verified that the maximum bandwidth directed spanning tree also gives the same trees as the ones shown in the figure.

The key insight we get from the example is that any tree creation algorithm cannot be oblivious of the network state left for subsequent iteration. Thus, any algorithm to create multiple trees has to try to leave as much bandwidth for future trees while trying to keep bandwidths of each of the trees as balanaced as possible.

C. Maximum Bandwidth Sum Tree (MBST) Algorithm

Our MBST algorithm incorporates the above insight in building the trees. It does so by checking the amount of outgoing bandwidth left at a node if one of its outgoing edge is added to the current arborescence. The MBST algorithm is shown in Figure 4.

The MBST algorithm starts by adding the source (say node 0) to the list of nodes that have been added to the current $InTree$ (line 4). Next, for each node k in the $InTree$ list, we find the maximum outgoing bandwidth arc to nodes outside the $InTree$ list (line 7). For each of node k already in the tree, we compute the leftover outgoing bandwidth E_k, if its maximum outgoing edge

Input:
N - Set of Nodes – S is the source
b_{ij} - Bandwidth between all pairs $i, j \in N$
E_n - Outgoing Access Capacity of $n \in N$

Output:
T - Set of Trees

Algorithm:
1. $T \leftarrow \phi$
2. Do
3. $CurrentTree \leftarrow \phi$
4. $InTree \leftarrow \{S\}$
5. While $InTree \neq N$ do
6. For each $n \in InTree$ do
7. $M_n \leftarrow$ Edge with $max_j\{b_{nj}\}$
8. $R_n \leftarrow E_n$ - {Bandwidth of M_n}
9. End For
10. $x \leftarrow$ Node with $max_j\{R_j\}$
11. $CurrentTree \leftarrow CurrentTree \cup M_x$
12. $InTree \leftarrow InTree \cup \{destination\ of\ M_x\}$
13. End While
14. For each edge e in $CurrentTree$
15. $b_e \leftarrow b_e$ - {bottleneck in $CurrentTree$}
16. End For
17. $T \leftarrow T \cup CurrentTree$
18. End Do
19. Return T

Fig. 4. Algorithm to create multiple trees (MBST)

is added to the current tree (line 8). The edge which leaves the maximum outgoing bandwidth for its source node, is added to the tree and its destination node is added to the $InTree$ list (lines 10–12). We repeat this process until all nodes have been added to the tree. Then, we reduce the bandwidth on all edges of the tree by the amount of bottleneck bandwidth (lines 14–16). We run this algorithm on the remaining graph until we cannot find any more trees. We note that the ability to use a single overlay edge in multiple trees is only available to us because of the sub-file multiplexing capability of the SPIDER architecture.

We trace the MBST algorithm on the example shown in Figure 3. The network graph is identical to that in Figure 2 except that its nodes are now labelled with their access bandwidth (E_s, E_1, E_2, E_3). For this example, we keep the values of the access bandwidths as the sum of the outgoing bandwidths from a link. The first edge $S \rightarrow N_3$ is added as shown in figure 3(B) as it has the maximum outgoing capacity (10 units). This reduces the outgoing capacity of the source (for future trees) by 10 units (thus $E_s = 19 - 10 = 9$). To add the next edge, we have two candidate edges $S \rightarrow N_1$ and $N_3 \rightarrow N_1$, which are the maximum capacity outgoing edges from the two nodes already in the tree. If we add $S \rightarrow N_1$ to the tree, then the outgoing capacity at source E_s would

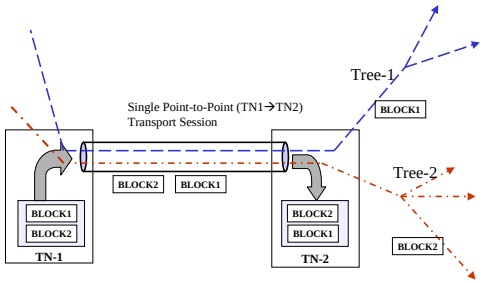

Fig. 5. Block transfer on a tree edge: blocks of different trees are transferred through a single TCP connection sequentially

be reduced to 1 (since the tree would have a bottleneck capacity of 9 and two edges out of S would be used, thus leaving 1 unit of spare capacity on $S \rightarrow N_1$). If we add $N_3 \rightarrow N_1$ to the tree, then E_1 would be reduced to 7. Since adding $N_3 \rightarrow N_1$ leaves more bandwidth at the source of the edge, we add it to the tree, despite knowing that $S \rightarrow N_1$ was more likely to create a larger bandwidth tree (Figure 3(C)). We update E_3 to 7 and also update E_s to 12 to account for the fact that 3 units of bandwidth on $S \rightarrow N_1$ would be available for future use. Using a similar procedure, we complete the first tree by adding $N_1 \rightarrow N_2$ to the tree. Figure 3(D) shows the complete first tree along with the updated outgoing access bandwidths at each node after the tree is created. We continue this process to get two more trees shown in figures 3(E) and 3(F). Note that the total capacity of these three trees is 16 units (7+7+2) whereas using SWPT we could only get 10 units of bandwidth. It could be verified that 16 units is the maximum transfer capacity one could extract in the above example.

D. Implementation note

In the SPIDER system, the MBST algorithm requires the knowledge of measured bandwidth among node pairs and the access bandwidth. In most CDN scenarios, various measurement infrastructure is already in place (e.g. Akamai as referred in [30]). In measuring the expected TCP throughput between two nodes, we use the following equation given in [31] as

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)},$$

where p is the loss event rate, s is the packet size, R is the round-trip time and t_{RTO} is the TCP retransmit value. The above sending rate measurement approach does not inject too much probe traffic.

III. Content Distribution using SPIDER

The SPIDER distribution system consists of three entities: source, Transit Nodes (TNs), and destinations.

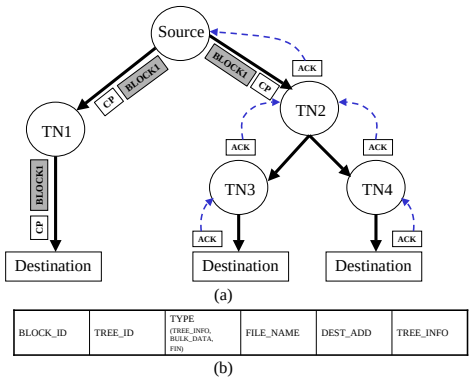

Fig. 6. a) Tree management has two parts: i) transmission of the blocks along the tree by sending a control packets preceding it; ii) ack propagation back to the tree ensuring reliable delivery of block. b) Control packet format.

The entire replication process is coordinated by the source and SPIDER agents installed at all destinations and TNs.

A. Block level transfer

At the source, the original data file is split into equal sized blocks. In our experiments and simulations we considered the size of the blocks as 512 KB. The size was chosen based on experimentation and is balance between large size to achieve high TCP throughput and low sizes for faster adaptation to bottleneck bandwidth fluctuation. For each block, the source decides which specific tree is to be used for the transfer from itself to the entire set of destinations. A block is transported using only a single source-specified tree. Each block is transferred hop-by-hop along the given tree using TCP or any point-to-point transport protocol as shown in Figure 5. When the block is completely transferred from transit node TN1 to the next hop TN2, TN2 replicates the block and transfers it to its children in the tree. The point-to-point transport session is terminated between two overlay nodes which essentially decouples the sending rate of two TCP sessions on any two tree edges - an approach also used in [25]. In order to avoid application-level buffer space overflow, we perform a window-based flow control at block level at the source and TNs (note that TNs may use a fast cache of finite size). Once the block reaches the intended destination, it is directly written into the original file.

B. Distribution tree setup

Each tree has a tree_id T_i. Each block is mapped to an unique tree_id at the time of transfer from the source node. The source sends a control packet before transmitting the block. The format of the control packet is shown in Figure 6(b). The fields of the control packet include a block_id (for identifying its position in the file and for acknowledgments) a tree_id, and also the topology of the tree (tree_info).

When a TN receives this control packet, it first adds the tree topology information in its *tree-table*, and subsequently forwards the control packet to its children on the tree as specified in the tree topology information (6 (a)). In this manner the control packet propagates to the entire tree prior to the block. This ensures that the tree forwarding information is always available at TN before the data block arrives. For each subsequent block that is to be distributed using the same tree, the tree topology information is not included in the control packet since it can be extracted by each TN from the tree-table maintained locally.

On receiving a block each child sends an acknowledgment back to the parent (6(a)). A node waits for ACKs from children before propagating the ACK to its parent node.

In a dynamic network environment such as Internet, it may be necessary to change the topologies of trees in the middle of the replication process in order to adapt to the changing bandwidth conditions. The proposed "tree-per-block" method allows trivial addition, deletion and tree re-configuration at the granularity of single block transfer time. Once the source has assigned a block to a specific tree (as described in the control packet) that block is distributed by using that tree alone. By keeping the granularity of block size adequately small (512 KB) in comparison to the total data volume ($>$ 64 MB) such a structure allows us to switch trees quickly based on changing network conditions.

C. Flow control at block level

When the source receives an ACK for a block from all its direct children, it is guaranteed that that the corresponding block has reached all the intended destinations. For each tree T_i, the source maintains the number of unacknowledged blocks b_i, which refers to the number of blocks that are in flight on the distribution tree. For a given tree T_i, there is a threshold W_i which acts as the buffer size at source. Blocks are pushed sequentially on tree i if $b_i \leq W_i$, i.e, there is space in the buffer. The W_i is chosen based on the maximum delay- bandwidth product for the tree i and can be time varying . The maximum delay can be found from time stamping the control packet and finding the time when it reaches all the destinations. The bandwidth can be computed from the ACKs.

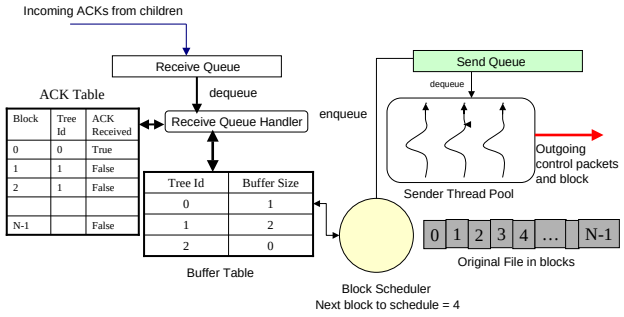

Fig. 7. Node level components

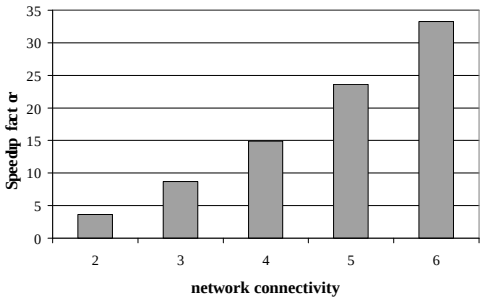

Fig. 8. Speedup of SPIDER over multi-unicast. (Simulations)

There are two reasons for having this window control. First, it ensures that the application buffer at TNs does not overflow while trying to maintain a non-zero buffer. The non-zero buffer is required to make sure that the TN always has data to send thus maximizing the link utilization. Second, this allows SPIDER to balance the load among trees. With infinite window, blocks will be sent on a tree based on the available bandwidth on the first hop and not on the bottleneck bandwidth of the trees. However, with the window, a tree which has more bandwidth will have ACKs coming at a higher rate increasing the rate of blocks being pushed in that tree. Without load balancing among the trees, the total throughput from the trees will not be maximized.

In certain cases, two trees may share the same overlay link. In the implementation of SPIDER, a single TCP session is opened on which blocks belonging to different trees can be transferred. This is done to avoid establishing multiple TCP connection between two nodes and thus ensuring fairness to other competing traffic. However, blocks may be assigned on the two trees at different rates, which is decided by the tree creation algorithm. In such cases the traffic on the single TCP connection (corresponding to the shared edge) is split between blocks assigned to the two trees proportionally based on the corresponding rates of the trees.

D. Resilience to Transit Node/Connection failures

In the unlikely event that a Transit Node fails, a parent wil discover through lack of acknowledgments and attempt to foward data directly to the childen of the failed node. In the worst case the affected block may need to be re-transmitted from the source, possibly using a new tree that will be automatically computed for subsequent blocks as the failure gets detected.

E. Node level architecture

The main components of basic node architecture (Figure 7) are: a) a Receive queue handler for processing control packets and ACKs, b) an ACK table for keeping track of delivered blocks, c) a buffer table used for flow control d) a scheduler for selecting trees on which to transfer a block and e) a Send queue to enqueue departing blocks on to a single TCP connection.

IV. Performance Evaluation and Experiments

In this section we present results from both simulation studies (to explain some properties of our proposed scheme) and detailed experimental results conducted on PlanetLab. In all these experiments we examine primarily two metrics of interest — the makespan of the replication process, defined by the last destination to complete the transfer, and net throughput achieved different destinations.

A. Simulation Studies

For our simulations we generate Internet-like topologies using Brite [36] of upto 2000 nodes. Typically 200 of these nodes serve as destinations for the replication process. Since we expect that the source, TNs and the destinations are part of a high-speed content delivery network, we assume that these nodes have relatively high bandwidth ($\sim$ 1 Mbps). We will examine more realistic scenarios in the PlanetLab experiments. For different experiments we generated such topologies with varying degree of network connectivity.

Comparison with multiple-unicast: We first compare the performance of SPIDER with a simple multi-unicast approach, in which the data is sent using multiple unicast transfers directly from the source to each destination. In Figure 8 we plot the speedup factor for the makespan, which we define as the ratio of the makespan for multi-unicast to the makespan for SPIDER. We can observe that in general the speedup gained by SPIDER in these experiments vary between 4 and 33 for different network

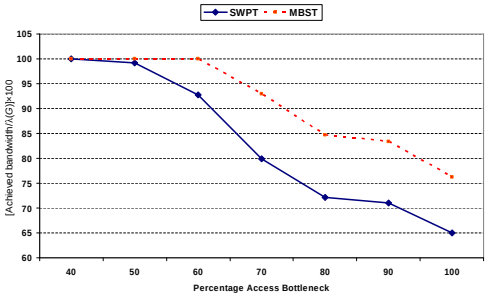

Fig. 9. Performance comparison of MBST and SWPT to an upper bound of the maximum bandwidth achievable, $\lambda(G)$. (simulations)

topologies, with greater speedup achieved in more connected networks. This is an expected behavior of SPIDER because increase in connectivity implies increase in path diversity which is efficiently exploited by SPIDER. In particular for network connectivity around 3-4 (as is true on the Internet) the speedup varies between 9-15.

Comparison of MBST and SWPT: In Section II we had compared two approaches for tree construction, namely SWPT and MBST, and had intuitively explained why the latter is a better choice. We new present experimental verification of this intuition. In Figure 9 we plot the variation of the total throughput achievable on all trees simultaneously, as the location of the bottleneck in the network is varied. (In these experiments we varied the bandwidths of the access links on the topology with respect to the rest of the links.) The X-axis of the plot shows the ratio of the average bandwidth of the access links to the average bandwidth in the rest of the network. Note that all links in the network are assigned one bandwidth value and all access links are assigned another bandwidth value. The y-axis plots the ratio $\beta/\lambda(G)$ as a percentage, where β is the sum of throughput achieved on all the trees tnd $\lambda(G)$ is the minimum of max flow values from the source to all destinations. In general we want a higher value of β, and as explained in Section II, $\lambda(G)$ is an upper-bound on β. We can observe that when the access links serve as bottlenecks (e.g., as access to network bandwidth ratio of 40%) the trees are primarily constrained by access bandwidths in both cases and they achieve the same performance as the minimum max-flow. Note that in such scenarios $\lambda(G)$ is also the optimal value for β. However, as the bottleneck shifts to the network (e.g., for access to network bandwidth ratio of 100%) our proposed MBST algorithm starts to perform better than SWPT, vindicating our choice. MBST achieves a total throughput within 76% of $\lambda(G)$ in such cases which SWPT achieves a throughout within 65% of $\lambda(G)$.

6test.edu.cn	China
bgu.ac.il	Israel
cs.princeton.edu	Princeton
cs.umd.edu	UMD
cs.vu.nl	Netherlands
cs.washington.edu	Washington
Millennium.Berkeley.EDU	Berkeley
cse.nd.edu	Notredame
ece.toronto.edu	Toronto
hpl.hp.com	HP
inria.fr	France
nbgisp.com	Intel
ie.cuhk.edu.hk	HK
cis.upenn.edu	UPENN
info.ucl.ac.be	Belgium
cs.ucla.edu	UCLA
diku.dk	Denmark
dcs.bbk.ac.uk	UK

Fig. 10. Planet-Lab sites and labels.

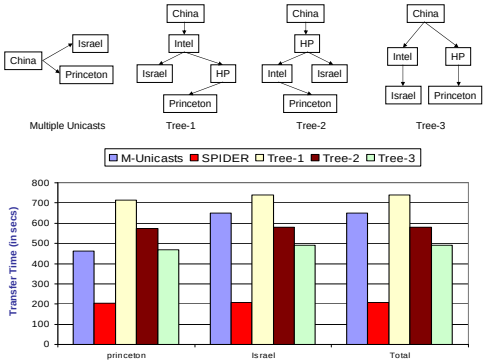

Fig. 11. Data Replication from China to Princeton and Israel using two TNs.

B. PlanetLab Experiments

We ran the PlanetLab experiments with upto 18 nodes (including source, TNs, and destinations) that were widely distributed in different sites. In order to avoid overloading of the shared PlanetLab nodes with our experimental traffic, we restricted our experiments to transfer files of size 64 MB from the single source to multiple destinations in all our experiments. In Figure 10 we list all the nodes that we have used in our experiments.

Comparison with multi-unicast and single-tree solutions: As a base case, we first compare the performance of SPIDER to simple multi-unicast and single tree based solutions with only two destinations and two other nodes as TNs. The source was located in China, the destinations were in Israel and Princeton, and the TNs were in Intel and HP (see labels in Figure 10). The top panel in Figure 11 illustrates the different trees that were

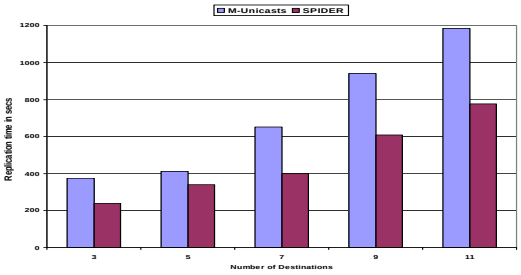

Fig. 12. Comparing transfer time with increase in the number of destinations.

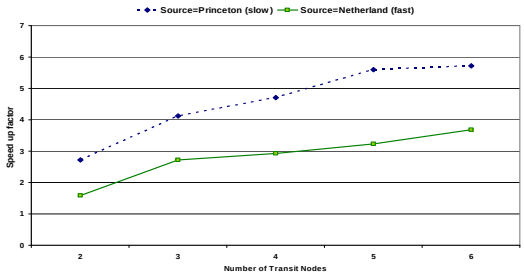

Fig. 13. Speed up factor with increase in TNs.

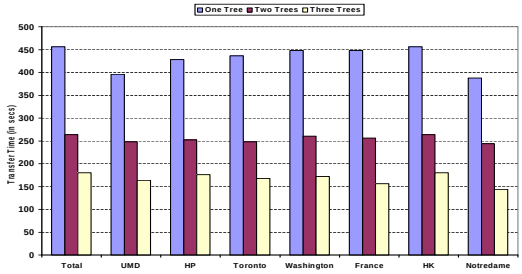

Fig. 14. Transfer time with increase in trees

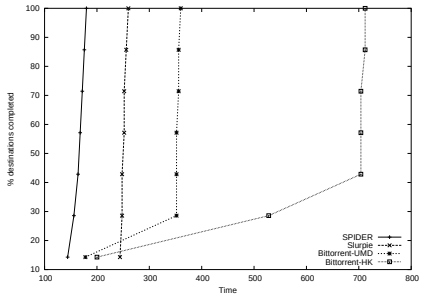

Fig. 15. Percentage completion of replication for SPIDER, BitTorrent and Slurpie.

generated. The leftmost tree was the multi-unicast tree (no TNs) and the remaining three topologies indicate the three trees that were created in SPIDER. In the lower panel of the figure we plot the replication latency to each of the two destinations as well as the makespan (marked as 'total') which is essentially the same as Israel in this case. The replication latency in case of multi-unicast (leftmost bar) was a factor of 2-3.5 greater than SPIDER (second bar from the left). The remaining three bars indicate the latency incurred if only a single tree was used for the replication process. In this example we choose the same three trees generated by SPIDER, but individually for this comparison. The advantage of using multiple trees in SPIDER is obvious from this plot. More importantly, replication performance on a single poorly chosen tree alone can be worse than multi-unicast.

In Figure 14 we perform a similar experiment with a total of 7 destinations, 4 TNs, and the source located in Netherlands and present the performance of SPIDER as the number of trees created is varied. We can observe that there is nearly a factor of 1.75 improvement in performance (makespan) when two trees is used instead of one, and a factor of 2.6 improvement in performance when three trees are used instead of one.

Varying number of destinations and TNs: In Figure 12 we examine the performance of the makespan metric in SPIDER as the number of destinations is varied (number of TNs constant (4), source at Netherlands, and number of trees used by SPIDER constant (3)). We observe that although the absolute performance of SPIDER keeps improving with increasing number of destinations (over multi-unicast), the relative improvement is about 1.75 in all these scenarios.

However, as the number of TNs was increased from 2 to 6, the opportunity for spatial diversity in SPIDER increased and led to improvements in relative performance with respect to multi-unicast, i.e., increasing speedups (Figure 13). This can be observed in the plot for experiments with different sources.

Comparison to other techniques – BitTorrent and Slurpie: We next compare performance of SPIDER to two different data download techniques, namely BitTorrent [8] and Slurpie [9]. The source code for both these schemes were publicly available. In order to keep the comparisons fair, we chose to run Slurpie in an "altruistic" mode, where each peer (in our case TNs and destinations) would stay in the system to assist other destinations in completing their downloads. In Figure 15 we compare the download latency of all seven destinations used in this experiment. The plot indicates the cumulative fraction of destinations that completed the download (replication) with time. We can observe

that destinations in SPIDER, which efficiently leverages multiple trees, finishes the entire replication process by 180 seconds, followed by Slurpie at about 260 seconds, i.e., an improvement factor of approximately 1.5. The performance of BitTorrent is highly variable and is sensitive to which destination is able to completely finish the first download. The lines marked BitTorrent-UMD and BitTorrent-France represents the best and the worst case scenarios for BitTorrent among all the different runs of the experiment (best case occured when the UMD destination was the first to finish download and the worst case occured when France was the first) Our results here also confirm the observations of authors in [9] that Slurpie achieves a factor of two improvement in download performance over BitTorrent.

V. Conclusions

In this paper, we have presented the SPIDER system for coordinated fast distribution of large content across multiple sites. SPIDER is suitable for replication in different infrastructure based systems such CDNs, scientific data exchange systems, and storage network. In SPIDER, data replication is accelerated by employing multiple trees rooted at sources that are computed through careful measurements. We present an algorithm for computation of multiple trees and discuss the SPIDER data transfer protocol. In our experimental evaluation of SPIDER on PlanetLab, we observe an improvement factor of 6 when compared to multi-unicast. In comparison to Bittorrent and Slurpie, SPIDER provides an improvement factor of 4 and 1.5 respectively.

References

[1] D. Patterson, "A Conversation with Jim Gray", *ACM Queue, vol. 1, no. 4*,June 2003.

[2] http://www.akamai.com/en/resources/pdf/casestudy/Akamai_CaseStudy_BMW_Films.pdf

[3] J. W. Byers, M. Luby and M. Mitzenmacher, "Accessing multiple mirror sites in parallel: using tornado codes to speed up downloads", *In Proc. of IEEE Infocom*,1999.

[4] J. Byers, J. Considine, M. Mitzenmacher and S. Rost, "Informed content delivery across adaptive overlay networks", *In Proc. of ACM SIGCOM*,2002.

[5] W. Cheng, C. Chou, L. Golubchik, S. Khuller and Y. Wan,"Large-scale data collection: a coordinated approach", *Proc. of IEEE Infocom 2003,* San Francisco.

[6] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, "SplitStream: High-bandwidth multicast in a cooperative environment," *In Proc. of SOSP 2003,* Lake Bolton, New York, October, 2003.

[7] D. Kostic, A. Rodriguez, J. Albrecht and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," *In Proc. of SOSP 2003,* Lake Bolton, New York, October, 2003.

[8] http://bitconjurer.org/BitTorrent/

[9] R. Sherwood, R. Braud and B. Bhattacharjee, "Slurpie: A Cooperative Bulk Data Transfer Protocol," *In Proc. of Infocom*, March 2004

[10] G. Kwon and J. Byers, "ROMA: Reliable Overlay Multicast Using Loosely Coupled TCP Connections," *In Proc. of IEEE Infocom*, Hong Kong, 2004.

[11] A. Akella, S. Seshan and A. Shaikh, "An Empirical Evaluation of Wide-Area Internet Bottlenecks," *In Proc. of IMC,* San Diego, CA, 2003.

[12] Y. Birk and D. Crupnicoff, "A multicast transmission schedule for scalable multi-rate distribution of bulk data using non-scalable erasure-correcting codes",*In Proc. of IEEE Infocom*, 2003.

[13] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker and J. Zahorjan, "Detour: a case for informed Internet routing and transport", *IEEE Micro*, vol(19), Jan 1999.

[14] D. Anderson, H. Balakrishnan, M. Kaashoek and R. Morris, "Resilient overlay networks", *In Proc. of 18th ACM SOSP,* Oct 2001.

[15] J. Jannotti, D. K. Gifford, K. L. Johnson, M.F.Kaashoek and J.W. O'Toole, Jr, "Overcast: reliable multicasting with an overlay network", *In Proc. of 4th USENIX OSDI Symposium*,pp.197-212.

[16] Y.H Chu, S. G. Rao and H. Zhang, "A case for end system multicast", *In Proc. of ACM Sigmetrics*, 2000.

[17] D. Pendarakis, S. Shi, D. Verma and M. Waldvogel, "ALMI: an application level multicast infrastructure", *In Proc. of 3rd Usenix Symposium on Internet Technologies & Systems (USITS 2001)*,San Francisco, March 2001.

[18] http://gridnets.nec-labs.com/

[19] C. Jin, D. Wei and S. Slow, "FAST TCP: motivation, architecture, algorithms, performance", *In Proc. of IEEE Infocom*, March 2004

[20] D. Katabi, M. Handley and C. Rohrs, " Congestion Control for High Bandwidth-Delay Product Networks, " *In Proc. of ACM Sigcomm*, 2002.

[21] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, S. Meder, S. Tuecke, "GridFTP Protocol Specification," *GGF GridFTP Working Group Document*, September 2002.

[22] E. He, J. Leigh, O. Yu and T. DeFanti, " Reliable Blast UDP: Predictable High Performance Bulk Data Transfer," *In Proc. of IEEE Cluster Computing*, 2002.

[23] J. Byers, M. Luby, M. Mitzenmacher and A. Rege, "A digital fountain approach to reliable distribution of bulk data", *In Proc. of ACM SIGCOMM*, Vancouver, Sept 1998.

[24] L. Cherkasova and J. Lee, "FastReplica: Efficient Large File Distribution within Content Delivery Networks", *In Proc. of 4th USENIX Symposium on Internet Technologies and Systems*,Seattle, Washington, March 26-28, 2003.

[25] G. Kola, T. Kosar and M. Livny, "A Fully Automated Fault-tolerant System for Distributed Video Processing and Off-site Replication," *In Proc of NOSSDAV,* June 2004

[26] K. Lai and M. Baker, "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay", *In Proc. of ACM SIGCOMM*,August 2000.

[27] J. Edmonds, "Edge-disjoint branchings", *Combinatorial Algorithms. Algorithmics Press, New York*, 1972, pp. 91-96.

[28] H. N. Gabow and K.S. Manu, "Packing algorithms for arborescences (and spanning trees) in capacitated graphs",*In Proc. of Math. Program 82*,pp. 83-109, 1998.

[29] K. Jain, M. Mahdian and M.R. Salavatipour, "Packing Steiner trees", *In Proc. of 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2003.

[30] A. Akella, S. Seshan, and A. Shaikh, "Towards Representative Internet Measurements," *In Proc. of NYMAN*, NY 2003

[31] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *In Proc. of ACM SIGCOMM*, Vancouver, 1998.

[32] M. Grotschel, L. Lovasz, A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*,Springer-Verlag, 1993.
[33] M. Charikar, C. Chekuri, To-yat Cheung, Z. Dai, A. Goel, S .Guha, M. Li, "Approximation Algorithms for Directed Steiner Problems", *In Proc. of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*,1998.
[34] R. Carr and S. Vempala, "Randomized metarounding", *Random Structures and Algorithms*,20(3): 343-352 (2002).
[35] U. Feige, "A Threshold of ln n for Approximating Set Cover" *JACM*,45(4): 634-652 (1998).
[36] A. Medina, A. Lakhina, I. Matta and J. Byers, "BRITE: Universal Topology Generation from a User's Perspective," *Technical report*,Dept. of Computer Science, Boston University BUCS-TR-2001-003, April,2001.
[37] M. Faloutsos, P. Faloutsos, C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *In Proc. of ACM SIGCOMM*,1999.

VI. Appendix: Approx-Algorithm for tree computation

We present an approach to design an approximation algorithm to solve the general case based on the corresponding problem in undirected graph studied in [29].

Based on previous definitions, our problem can be stated as a linear programming problem:

$$\max\left\{\sum_{T\in\mathcal{T}} b(T) : \sum_{T:T\ni e} b(T) \le c_e \forall e \in E, b(T) \ge 0 \forall T \in \mathcal{T}\right\}.$$

In other words, it requires an assignment of non-negative capacities (denoted $b(T)$) to the directed tree T from source s connecting D in a set of all such trees $\mathcal{T}$, such that the sum of these capacities is the maximum possible under the constraint that for any edge e, the sum of the capacities of the trees containing e is at most the capacity c_e of e. This LP problem has exponentially many constraints and variables and standard LP algorithm will require exponential amount of time. The dual problem of is more tractable:

$$\min\left\{\sum_{e\in E} c_e y_e : \sum_{e\in A} y_e \ge 1,\ \forall A \in \mathcal{A}, y_e \ge 0\right\}.$$

In other words, the dual problem associates a non-negative real number y_e with each edge such that a weighted sum of these variables is the minimum possible and for every directed tree T rooted at s spanning D in $\mathcal{T}$, the sum of y_e for e an edge in T is at least 1.

This problem has polynomially many variables although the number of constraints is still exponential. The ellipsoid algorithm [32] can be used to solve it, i.e., provided that we have a polynomial algorithm to check if a proposed solution is feasible; and if not, find a violated constraint. Testing feasibility still remains a hard problem as it reduces to the NP-Hard directed Steiner tree problem. An α-approximation algorithm for this problem leads to an α-approximation algorithm A for our original packing problem. We briefly describe the proof of this based on [29].

To solve the optimization problem, as in the above dual linear program, we add the inequality $\sum_{e\in E} c_e y_e \le R$ to the set of constraints in the dual and check if the resulting linear program is feasible using the ellipsoid method. By doing binary search on R, we can get arbitrarily close to the true optimum. Each step of the ellipsoid algorithm maintains y_e's, and checks whether the present y_e's form a feasible solution. In our case, for this checking the ellipsoid algorithm uses A. If A finds a tree with weight less than 1, that means that the program is infeasible. If it does not, then we cannot be sure whether the program is feasible or not, but we do know that if we replace R by αR then αy_e's form a feasible solution because A is an α-approximation algorithm. This implies that if the ellipsoid algorithm finds the solution to the dual to be R^*, then the actual solution to the dual, and hence to the primal lies between R^* and αR^*. Thus we get an α-approximate algorithm for our packing problem.

In [33] an approximation algorithm is given for the minimum directed Steiner tree problem, which finds a directed Steiner tree with weight at most k^ϵ times the weight of the minimum weight Steiner tree for any fixed positive ϵ and with polynomial running time (k is the number of Steiner vertices). This gives us a polynomial time algorithm to find a fractional packing of the trees with weight at least $k^{-\epsilon}$ times the weight of the maximum weight fractional packing. As this algorithm only gives an approximate value for the dual but not for the primal, we need the solution for the primal in order to find the packing. To obtain the actual approximate packing, we can use the techniques of [34]. In solving the dual approximately, the ellipsoid algorithm finds polynomially many separating hyperplanes which show that the solution of the dual is at least R^*. So if we set the variables in the primal which correspond to constraints other than those which arise from the separating hyperplanes, then the resulting program has polynomial size and still has solution $\ge R^*$. The solution to this program gives us the desired packing.

Authors in [33] observe (using the hardness result of [35]), one cannot approximate the directed Steiner tree within a factor smaller than $\ln k$ in polynomial time implying that it is hard to find a tree packing with weight more than $1/\ln k$ times the weight of the maximum weight packing.