

Efficient Peer-to-Peer Content Distribution

Danny Bickson, Danny Dolev and Yair Weiss
 School of Computer Science and Engineering,
 The Hebrew University of Jerusalem.
 Email: {daniel51,dolev,yweiss}@cs.huji.ac.il

Abstract— We consider the problem of distributing data to a large and dynamic network of nodes. We formalize the *next-step* problem, which is intuitively the next action that each peer in the content distribution network should take, where an *action* is the transfer of one file part from a neighboring node.

We formulate the next-step problem as an integer programming problem and use a probabilistic graphical model in order to find the most probable network state using inference. Each node takes the most probable action calculated locally. For running inference, we use the max-product belief propagation algorithm, a decentralized message-passing algorithm.

Our paper makes two novel contributions: First, as far as we know, we are the first to propose a distributed algorithm to coordinate and to plan ahead the file parts replication across the network. Second, we propose to model the node selection and chunk selection problems using a probabilistic graphical model and to find a good approximations to these problems using a distributed inference algorithm. The initial assignment of probabilities to each action allows flexibility to support different dissemination policies, i.e. real-time streaming and bulk-data transfer.

Using extensive simulations we verify the performance of our algorithm relative to state-of-the-art deployed solutions.

I. INTRODUCTION

The *Content Distribution* problem, aimed at the concurrent distribution of bulk-data to a large and dynamic group of clients, is a problem that will need to be solved in the future of the Internet. The Content Distribution problem is sometimes called *application level multicast*. This is to differentiate the problem from an IP multicast, which is commonly supported at the network level. This problem has received a large amount of research attention in recent years. A naive solution to this problem involves unicasting the data to individual clients from the source node; However, this approach is not scalable. Another simple solution is server replication, a solution that might not be cost effective for all users.

A large fraction of the Internet bandwidth is used by P2P file sharing applications. File sharing networks like KaZaA, eMule [18] and BitTorrent [7], which have

millions of connected users online. Most of the solutions can be categorized based on their topology: Tree-based solutions, which create a hierarchy of clusters, are proposed in [11], [5], [1]. Meshes of multiple trees are proposed in [21], [4]. Unstructured meshes are used in BitTorrent [7].

In most of the existing Peer-to-peer solutions, the file is divided into parts and the nodes exchange these parts until they re-assemble back to the complete file. The solutions can be divided roughly into two sets: solution based on structured topology, and solutions based on unstructured topology.

In solutions based on structured topology, first the topology is constructed and then the file parts are disseminated along the routing paths defined by the topology. A drawback of many solutions based on structured topology is that they require constant maintenance of the topology and thus they are failure prone.

The other family of solutions is based on unstructured topologies. Usually those solutions optimize the local selection of each routing path used for dissemination of file parts. For example, in the BitTorrent system each node optimizes its download bandwidth to a fixed set of nodes. However, the nodes are not coordinated, a situation that might lead to oscillations: many nodes select the node with highest upload bandwidth, making it congested, then selecting the second highest upload bandwidth node and so on.

It is surprising that no current solution proposes to run a distributed algorithm for optimizing the file's parts replication ahead of the actual replication process. On the assumption the replicated files are large, the replication might take several hours and even days until completion. It is clear that in the existing solutions the data transfer is not optimized globally, and there are still many challenges to face in this direction and a significant room for improvement.

We formalize the *next-step* problem, which is intuitively the next action that each peer in the content distribution network should take, where an *action* is the transfer of one file part from a neighboring node. We for-

mulate the *next-step* problem as an integer programming problem and use a probabilistic graphical model in order to find the most probable network state using inference. Each node takes the most probable action calculated locally. For running inference, we use the max-product belief propagation algorithm, a completely decentralized message-passing algorithm, suitable for a Peer-to-peer environment.

The novelties in our approach are twofold: first, as far as we know, we are the first work which proposes to optimize the file part replication across the network before the actual download process. Second, we utilize algorithms from the domain of probabilistic graphical models for solving the content distribution problem distributively. This is done by first formalizing inference problem, and then solving it using the belief propagation inference algorithm. This approach has several appealing properties: the executed algorithm is very simple to implement, requires only local knowledge, does not use any predefined network topology, is scalable to very large networks, can operate in asynchronous and dynamic settings and handles failures well. Furthermore, our solution is modular in the sense that any distributed inference algorithm can be used instead of the belief propagation algorithm based on the topology settings. We allow multiple initialization policies for the inference algorithm, supporting for real-time streaming or bulk-data transfer and demonstrate using simulations that our proposed algorithm achieves good performance results relative to the existing greedy algorithms.

We use several simplifying assumptions about the network model, as do most of the recent works on content distribution modeling [17], [19]: We assume that the network is synchronous and we assume that nodes receive one file part and send one file part on each round. However, it is important to note that these assumptions simplify only the explanation of the algorithm and the implementation of the simulation: in practice these assumptions can be relaxed. Specifically, there is no need for synchronization of the nodes. In Section V we discuss how to adapt our theoretical algorithm to actual Peer-to-peer networks which do not have these constraints.

The remainder of this paper is organized according to the following format. In Section II we present the content distribution problem in general and the *next-step* problem in particular. The use of probabilistic graphical models for modeling content distribution networks is discussed in Section III. In section IV we propose an algorithm for solving the next-step problem, based on the probabilistic graphical model. We discuss the algorithm's properties and their applications to real world Peer-to-

peer networks in Section V. A performance evaluation of the algorithm, based on extensive simulations, appears in Section VI. An extension to support streaming media is presented in Section VII. In Section VIII we discuss algorithm performance when facing a simple failure model. Finally, we refer to related work in Section IX.

II. STATEMENT OF PROBLEM

Informally, let's assume a source node s obtains a complete file that needs to be distributed to all target nodes. A group of n target nodes is connected through an undirected graph topology. For the download to be efficient, the source node partitions the file into k equally-sized parts. Each edge in the communication graph is associated with a non-negative link cost that defines the cost of transferring one part of the file via that edge. For simplicity, we assume that the communication is synchronous, meaning that the nodes operate in rounds. During each round, every node can send, at most, one part of the file and likewise receive at most one part from its neighbors. This model is called a *send/receive model* or a *1-port model* in the parallel computing literature. We ignore link latencies and assume that a file part, which is sent in round r , arrives at its destination at the same round.

One can optimize the download according to various performance measures. One such measure could be the download time until the last part arrives at the last node. Another measure could be the *communication cost*, which is defined as the product of the number of file parts traversing each link, by that link's cost, summed up over all links. This measure is also called *resource usage* [5].

A. Next-step Problem

Given a communication graph $G = (V, E)$ where $|V| = n$, a weight function $W : V \times V \rightarrow \mathcal{R}$ which gives each graph edge a real non-negative weight, a file F that is divided into k parts, and an $n \times k$ binary matrix A :

$$A_{ij} = \begin{cases} 1 & \text{if node } i \text{ has file part } j \\ 0 & \text{otherwise.} \end{cases}$$

The *next-step* problem involves the calculation of the actions of each node, represented by an *action matrix* R of size $n \times k$,

$$R_{ij} = \begin{cases} k & \text{if node } i \text{ takes part } k \\ & \text{from node } j \\ 0 & \text{otherwise,} \end{cases}$$

subjected to the following constraints:

- 1) if $R_{ij} = k$ then $A_{ik} = 0 \wedge A_{jk} = 1 \wedge e_{ij} \in E$;

- 2) if $R_{ij} = k$ then $\forall t \neq i, l \neq j R_{tj} = 0 \wedge R_{il} = 0$;
- 3) $C(R) = \min_{R' \in R} C(R')$

Constraint (1) implies that if a node takes a part k from node j , then three conditions must be met: Node j has to have part k , node i must be missing part k and node i and j must share a common edge e_{ij} . Constraint (2) means that each node can send, at most, one part and each node can receive, at most, one part in each communication round. We would like to find the minimal cost solution, where $C(\cdot)$ is some non-negative cost function which evaluates the solution (Constraint 3). Next, we define the cost function we use in this paper.

Given a solution to the *next-step* problem, the cost function assigns it a real non-negative cost. Intuitively, we would like to minimize this cost. In this paper we use the following cost function:

$$C(R) = \frac{1}{\epsilon} \sum_{ij} \delta_{ij} + \sum_{R_{ij} > 0} w_{ij} + \sum_i w_{ii}$$

Where δ_{ij} is defined to be:

$$\delta_{ij} = \begin{cases} 1 & \text{If nodes } i \text{ and } j \text{ take a common part} \\ & \text{from the same neighboring node } k \\ 0 & \text{otherwise.} \end{cases}$$

Note that if δ_{ij} is one, we assign an infinite cost to it using the term $\frac{1}{\epsilon} \sum_{ij} \delta_{ij}$. The second term of the cost function is the sum of edge costs w_{ij} : this is the cost for transferring a part via the graph edge e_{ij} . The third term is the sum of node and parts costs: w_{ii} is composed of the cost of using the node i and the cost of transferring a specific file part. In the next subsection we give a detailed example how those costs are calculated.

The action matrix reflects a possible exchange of parts among the nodes in a given round. It is important to note that the action matrix is defined in such a way that once we reach the *final state* in which $\forall i, j A_{ij} = 1$, the only possible action matrix is the “0” matrix.

B. Content Distribution Problem

Given a graph $G = (V, E)$, a source node $s \in V$ and a cost function $C(\cdot)$ which assigns a non-negative cost to each action matrix R , cost, the Content Distribution Problem seeks to determine a sequence of action matrices which reach the final state, while minimizing the accumulated cost. Thus, the initial matrix $A_{ij}^{(1)}$, (representative of the state where the single source node s has the complete file), is

$$A_{ij}^{(1)} = \begin{cases} 1 & i = s \\ 0 & \text{otherwise.} \end{cases}$$

The selected sequence of m action matrices, $R^{(1)}, R^{(2)} \dots R^{(m)}$, should satisfy the following conditions:

- 1) m is the minimal round number for which all nodes have finished the download of all file parts;
- 2) $R_{ij}^{(t)} : A_{ij}^{(t)} \rightarrow A_{ij}^{(t+1)}$, is a solution of the next-step problem; and
- 3) The total cost, $\sum_{l=1}^m C(R^{(l)})$, is minimal.

III. GRAPHICAL MODELS FOR CONTENT DISTRIBUTION

An undirected graphical model G consists of a set of vertices V and a set of edges E connecting them. Each vertex v_i is associated with a random variable x_i . We assume that the probability distribution $p(x)$ factorizes into a product of terms involving node pairs and single nodes. These factors are called edge potentials $\psi_{ij}(x_i, x_j)$ and local (or self) potentials $\psi_{ii}(x_i)$.

The max-product belief propagation algorithm [20] is a distributed inference algorithm that enables us to calculate the MAP (maximum a posteriori) assignment of the nodes, otherwise known as the “beliefs.” It is a distributed message-passing algorithm and is therefore suitable for communication networks [8], [14], [22]. The Belief Propagation (BP) algorithm gives exact results on trees. We have no guarantee for the algorithm performance on graphs with cycles. The input to the BP algorithm is a graphical model with self potentials $\psi_{ii}(x_i)$ and edge potentials $\psi_{ij}(x_i, x_j)$. The output of the algorithm is the vector of node beliefs (posteriori probabilities). The algorithm is an iterative distributed message passing algorithm where messages sent between nodes are determined by the following update rule:

$$m_{ij}(x_j)^{(t+1)} = \alpha \max_{x_i} \psi_{ij}(x_i, x_j) \psi_{ii}(x_i) \prod_{x_k \in N(x_i) - x_j} m_{ki}^{(t)}(x_i) \quad (1)$$

where $m_{ij}(x_j)$ is a message sent from node x_i to node x_j , α is a normalization factor and $N(x_i)$ is the set of neighbors of node x_i . We initialize the messages at the first round uniformly. Finally each node calculates the belief:

$$bel(x_i) = \alpha \psi_{ii}(x_i) \prod_{x_j \in N(x_i)} m_{ji}(x_i) \quad (2)$$

The algorithm converges when the node receives identical messages from all neighbors for two consecutive rounds. In networks containing cycles the algorithm might not converge. However, in practice, there are several applications where the algorithm produces very

good results even for graphs with cycles, for example, in the case of Turbo codes.

It is known that if we use the max-product algorithm we can find an optimal X^* that maximizes the probability $P(x)$, if the topology has no cycles, and we can find a strong local maximum in case the graph has cycles [28]. In other words, our algorithm solves the problem optimally on trees and gives a good approximation of a graph containing cycles. Based on equation 1, we calculate the self potentials $\psi_{ii}(x_i)$ and edge potentials $\psi_{ij}(x_i, x_j)$ for a given cost function. A future work might be to replace the belief propagation algorithm with other inference algorithms and to test their optimality on different topologies.

IV. PROPOSED ALGORITHM

We use a graphical model to represent the content distribution network. In our model, the nodes participating in the download are represented by vertices and the communication links to direct neighbors are represented by the graph edges. In this section we explain how to translate the next-hop problem into a graphical model formulation, solve the problem using the BP max-product inference algorithm, and use the MAP assignment results as a solution to the next-hop problem.

Each node has only local knowledge: a bitmap of the parts which it presently holds, and the list of its direct neighbors and the parts they hold. For each node k we construct an action matrix $B(v_k)$ of size $n \times k$

$$B_{ij}(x_k) = \begin{cases} 1 & \text{if node } k \text{ can potentially} \\ & \text{take part } j \text{ from node } i \\ 0 & \text{otherwise,} \end{cases}$$

The action matrix $B(v_k)$ lists all the possible actions of node k . Our goal is to decide which action out of $B(v_k)$ is the most profitable action to take. For solving this problem, we assign each node v_i a hidden random variable x_i which is assigned one of the possible actions in $B(v_i)$.¹

The goal of running the max-product BP algorithm is to find a MAP assignment of values for the hidden variables x_i that will minimize the cost of a the solution for the next-step problem. The output of the BP algorithm is the assignment of each random variable x_i with one possible action of node i .

For running the BP algorithm we need to assign for each node the *self potentials*, noted $\psi_{ii}(x_i)$, the initial

¹Note that for simplicity we translate the matrix $B(v_i)$ into a vector of possible matching actions, where actions that are not possible are ignored.

probability of each possible action out of the possible action matrix $B(v_i)$.

For constraining the node's actions, we use the following graph construction. We add a constraint edge e_{ij} between any two nodes which can take a part from a common neighboring node. We call this constraint an *edge potential*, noted $\psi_{ij}(x_i, x_j)$. This is a table composed of the Cartesian product of two nodes' v_i and v_j actions. It is a pairwise probability distribution which gives a certain probability for each intersection of the two matching actions. In this table, we will assign a very small value, ϵ , to any intersection of the two nodes v_i, v_j actions that involves taking a part from the same third node v_k . We set the local potential $\psi_{ii}(x_i)$ to be $e^{-C(x_i)}$ where $C(x_i)$ is the cost of the action taken by node v_i .

After constructing the self potentials and edge potentials, we are able to run the inference algorithm to calculate nodes' beliefs. The algorithm decides locally which action each node should take by choosing the most probable action based on those beliefs.

It is known that if we use the max-product algorithm we can find an optimal X^* to maximize the probability $P(x)$, in case the graph does not have cycles - and a strong local maximum in case the graph does have cycles [28]. In other words, our algorithm solves the next-step problem optimally on trees and gives a good approximation for a graph containing loops.

For the simplicity of explanation we initially assume that download process is composed of rounds and in each round the inference algorithm is re-run (we relax this assumption later on). In each round every node can send and receive one file part. We assume at the beginning of the download that one graph node is the *source* node that contains all the file parts and that the other graph nodes have no parts of the file. In practice, however, there is no need for synchronous settings; recent works (e.g. [14]) show that the BP algorithm performs well in asynchronous settings as well.

It is clear that the source node is able to serve only one other node on the first round, since we assume that each node can send only one part in a round. On the second round, at most two parts can be propagated in the network: one from the source node, and another from the node that received the part in round one. We continue to propagate parts along the graph edges, until all nodes receive all file parts.

A. Calculating Self Potentials

In the previous section, we derived the self potentials $\psi_{ii}(x_i)$ from the cost function. We now provide a

detailed example to demonstrate the method in which these values are calculated. For simplicity of explanation, we use rational numbers (such as $1/2, 1/3, 1/4$) in the assignment of the probabilities. (In the general case we use $e^{-c(x)}$.)

We define a node's *action* as the possible transfer of a part to the node from a neighboring node. A special case is a *source* node, which has no actions to take: since it holds all parts at the outset. For example, consider the network in Figure 1: It has $n = 4$ nodes, and the number of file parts is $k = 3$. Under each node, the bitmap of its file parts is shown. Node C is a source node since it has all three parts; the other nodes have some parts of the file missing. Node B, for example, has 4 possible actions to take (i.e. parts to receive): part 3 from A, 2 from C, 3 from C and 3 from D.

We now assign a probability distribution for each of these possible actions. We have experimented with several existing policies for assigning these distributions. The first such policy is a uniform assignment of values. In our example, this would be: $\psi_{BB}(x_B) = (1/4, 1/4, 1/4, 1/4)$ - or, in other words, node B can take any of the four actions with equal probability. The *rarest part first* policy assigns the highest probability to the part that has the lowest frequency in the network. In our example, such a policy would assign the probabilities: $\psi_{BB}(x_B) = (1/6, 1/2, 1/6, 1/6)$ - the highest probability is $1/2$, since part 2 has only one occurrence in the network, when the other parts occur 3 times. Intuitively, it is better for a node to take a part needed by many other nodes need than to take a part which occurs frequently; such an approach is also taken in [7], [3]. Lastly, the *work* policy takes a part from the closest node, where distance is defined as being inversely proportional to the edge cost. Note that the number of actions could be different for each node, depending on the states of the node and its neighboring nodes. In the table below,

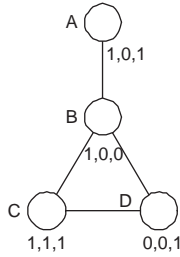


Fig. 1. Example network of four nodes with some initial state. Under each node, the obtained parts is shown.

the self potential of node B, (the initial probability to take a part from another node) ψ_{BB} is calculated using

two possible policies: Uniform and Rarest part first. The column headers are the 4 possible actions for node B, and the rows show the different policies. For example "1 from A" means that node B can take part 1 from A with probability $1/4$.

$\psi_{BB}(X_B)$	3 from A	1 from C	3 from C	3 from D
Uniform policy	1/4	1/4	1/4	1/4
Rarest part first	1/6	1/2	1/6	1/6

Table 2. Possible actions for the node B in the example graph shown in Figure 1

B. Graph construction of edge potentials

In the send/receive model, each node can send one part and receive one part per round. Thus, no two nodes can take a part from their mutual neighbor in the same round. For example, in Figure 2, nodes D and B are both neighbors of C, and can potentially take a part from C. We would like to minimize the possibility of this happening at the same round. In order to force such a constraint, we add an edge to the graph from node B to node D; In the pairwise probability table by assigning an ϵ to every intersection of actions of B and D, in which both take from the node C (table 3). In general, we apply this constraint for every two nodes, which might possible take a part from some common neighbor.

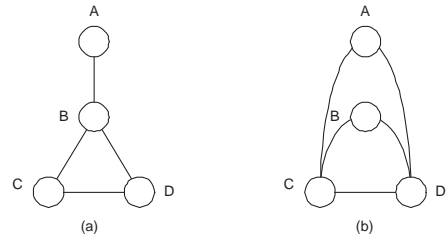


Fig. 2. (a) Example network topology (b) same topology after adding constraint edges and removing original edges. In the table below, the edge potentials ψ_{BD} is shown for the edge B-D. The table is a Cartesian product of the possible actions of nodes B and D, where there is an negligible probability (epsilon) for both nodes B and D of taking a part from a common node (node C in the example).

$\psi_{BD}(X_B, X_D)$	3 from A	2 from C	3 from C	3 from D
1 from B	1/8	1/8	1/8	1/8
2 from C	1/2	ϵ	ϵ	1/2
3 from C	1/2	ϵ	ϵ	1/2

Table 3. Example of the edge potentials for the edge BD for the graph shown in Figure 2. The matrix rows are then normalized.

C. Running the Loopy BP algorithm and processing the results

After constructing the self and edge potentials, we are ready to run the Loopy BP algorithm. Since we might have loops in the graph, the algorithm may not converge, and so we have to limit the number of algorithm iterations.² The output of the BP algorithm is the node's *beliefs*: A node belief is a vector with length equal to that of the self potential vector, with each entry equal to the probability of the matching action. The node selects the action with the highest probability from this belief vector; in case there are several equal probabilities in the vector, the node randomly chooses from among them.

Table 4 describes the results of running the Loopy BP algorithm on the example network. The upper two rows show the self potentials of the nodes B and D before the running of the algorithm; the lower two rows show the beliefs of nodes B and D. In this run, both B and D initially wanted to take part 2 - the rarest part - from C. However, because of the constraint edge BD, node D gave up taking a part from C and decided to take a part from B instead.

ψ_{BB}	0.166 (3A)	0.500 (2C)	0.166 (3C)	0.166 (3D)
ψ_{DD}	0.200 (1A)	0.200 (1B)	0.600 (2C)	
bel(B)	0.162 (3A)	0.488 (2C)	0.174 (3C)	0.174 (3D)
bel(D)	0.080 (1A)	0.677 (1B)	0.244 (2C)	

Table 4. Running the Loopy BP algorithm on the example network shows the results for nodes B and D. Notice that the initial probability of taking part 2 from C was 0.500 and 0.600 in B and D, since it is the rarest part in the network. However, since we have added a constraint edge B-D, node D gave up the action of taking 2 from C, and decided to take 1 from B. The highest probability actions are shown in bold.

V. ALGORITHM PROPERTIES

An immediate result of the construction described in equation ??, is that our algorithm solves the *next-step* problem optimally on tree topologies, while also giving a good approximation of an optimal solution on graphs containing loops.

In Section VI we present experimental results which show that our algorithm improves both running time and communication costs, in comparison to the greedy algorithm. However, we pay for this in extra communication, computation and time overhead. Next, we analyze the above costs and propose practical ways to adapt it to real world applications.

Regarding computation, note that the BP algorithm is composed only of linear vector and matrix multiplications. Performing these computations can be done

²see next session for a discussion on the feasibility of such an approach

even with weaker computing platforms such as sensor networks [8], [22].

With respect to the *time overhead*, we assume is that the algorithm computations are done in the background, while the content is being downloaded anyway. Recent performance analysis of P2P file sharing networks like BitTorrent, KaZaA and eMule [15], [23], [10], [13] show that download sessions may take several days until completion. For example a study [2] of the the BitTorrent network reported that the median file size is 600Mb and the mean session length is 13 hours. Another fact is that users experience idle time while waiting for their neighbors to be ready for uploading. Given this, one can start downloading using the greedy approach (as done in BitTorrent), while calculating better future actions in the background.

Without filtering out low probability actions in the BP algorithm, the communication overhead can be exponential. The communication overhead depends on many parameters: among them are the node degree, the number of algorithm iterations, the number of possible actions for each node (which depends on the number of file parts), and the network diameter. A realistic example setting of 20 neighbors and 100 file parts would require the sending of messages of size 20x100 to the neighbors and further require 100 BP algorithm iterations. Thus, we could easily get a communication cost overhead of 4Mb per file part per node. Luckily, we can filter out most of a node's actions and keep only a constant number of actions, specifically, those with high probability - with a very minimal effect on the algorithm performance. One can explain this since we are keeping the most probable actions out of all possible actions, and ignoring those actions that have negligible probability. In table V, we demonstrate the protocol overhead per chunk of information after applying the filtering: as the chunk size grows the overhead is reduced. For a part size of 256Kb, such as used in BitTorrent [7], the protocol overhead is a little below 1%. For part size of 9.1Mb, as used in eMule [18], the overhead is less then 0.02 percent.

Chunk size	Protocol Overhead
256Kb	0.93%
512Kb	0.46%
1Mb	0.23%
2Mb	0.11%
9.1Mb	0.02%

Table 5. Protocol overhead per chunk size per node

There are some open questions that need to be addressed further. The belief propagation algorithm is exact on trees and polytrees and the running time is the diameter of the tree, but is not guaranteed to converge on

graphs which contain cycles. In practice, however, it has nevertheless demonstrated very impressive performance in problems that contain cycles [8]. In communication graphs containing cycles, we limit the algorithm to a fixed number of iterations. As shown in the next section - this improves results - even when there is no convergence; however, the BP algorithm may fluctuate before converging into some local minima. A possible solution is to replace the BP algorithm with another algorithm that is known to converge [26]: our scheme, allows the use of any inference algorithm. Deciding on the best inference algorithm for a given network topology and cost function, is an area for future research.

VI. EXPERIMENTAL RESULTS

To test our algorithm, we implemented a simple packet-level discrete event simulator in Java, consisting of 3000 lines of code. All nodes in this simulator are synchronized and operate in rounds. In each round, we run our next step algorithm to decide which action should be taken by each node. As a reference we use the greedy algorithm, which for each node takes the action with the highest probability, without any communication between the neighbors. We used two kinds of graphs, both generated by the Georgia Tech topology generator (GT-ITM) [9]. These two graphs included a random graph, and a transit-stub graph of up to 600 nodes. We used the routing policy weights, also generated by the Georgia Tech topology generator, for assigning edge weights used for calculating the communication cost. Each simulation was repeated at least 10 times and the results were averaged.

At the beginning of each simulation, a random graph node is chosen to be the source node. All other nodes are initialized to have no parts. The source node starts to disseminate the file parts such that, in each round, it sends one part to one of its neighbors. The decision of which action to take on each part is done distributively, using our next-step algorithm: thus, our algorithm runs at every round.

An important question is how the ability to communicate and to load-balance neighboring nodes improves the total network performance. In Figure 3 we see the improvement in running time of our algorithm relative to the greedy algorithm: The x-axis represents the finished machines (sorted by finish times) and the y-axis represents the finishing time in terms of the number of simulation rounds. We can clearly see that the performance improvement in running time is about 20%. One of the advantages of our algorithm is that we do not define what incentive policies should be used

to initialize the self potentials required to run the BP algorithm. This adds flexibility for supporting numerous policies in maximizing different utility functions. We compared several different policies using - as before - a GT-ITM network of 600 nodes, with one node randomly selected to be the source node. We experimented with several policies: a uniform policy, in which each node selects a possible action at random; a rarest part first policy, in which the rarest parts have a higher initial probability; and a work policy, in which links with lower communication costs have a higher probability. All of the above three policies are known and widely used in content distribution networks. However, our scheme allows the combination of policies. Since the policies are probability vectors, one can pick any subset of policies, multiply them using dot product multiplication and then normalize the result. This discussion about the combination of policies is not only of theoretical interest. As shown in Figure 3, a combined policy of the rarest part+work policy achieved the best finishing time. An explanation to this might be that rarest part policy improves the heterogeneity of parts in the network while the work improves the link utilization.

Figure 4 plots the average progress of nodes per round for the above experiment, where progress is defined as the ratio of the number of nodes that were satisfied in the indicated round and the total number of unfinished nodes. The greedy algorithm is inferior to our algorithm in terms of progress per round. An interesting question that arises is how to characterize of the trade-off between download speed and download cost. Intuitively, both goals are contradictory: On the one hand, if we want to minimize the download time we are able to use high cost links to increase the network throughput. On the other hand, minimizing communication costs could cause time delays incurred while waiting for low cost links to become available. Surprisingly, by using a combined policy of work+rarest, we show that it is possible to minimize both the download time and the communication cost, as shown in Figure 5 (where the x-axis represents round numbers and the y-axis represents the cumulative communication cost).

In another experiment, we studied the effect of file size on the download performance; the results are shown in Figure 6. In this experiment, we assume that all file parts are of the same size - in other words, more parts indicate larger files. We experimented using file sizes of 20, 40 and 80 parts. The worst download time in rounds of a file of 20 parts was approximately 200 rounds. If we were to assume a linear increase in performance, we could assume that the worst finish time for 80 parts

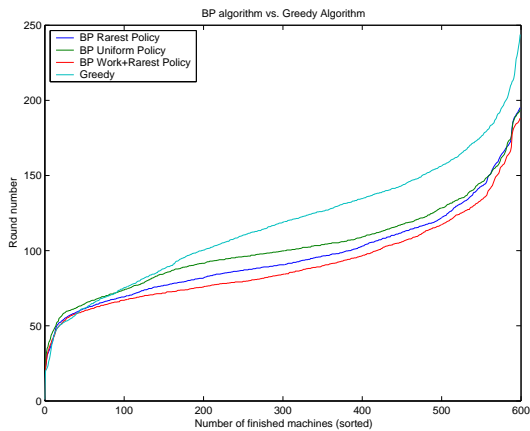


Fig. 3. Finish times for 600 nodes for a GT-ITM network. The policies used are uniform, rarest part first, work combined with rarest part first vs. the greedy algorithm.

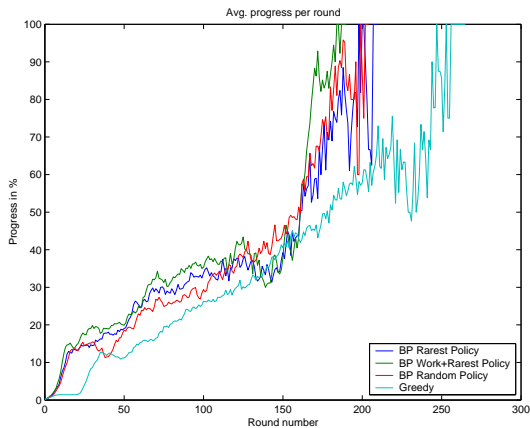


Fig. 4. Average progress per round for the same experiment in Figure 3. The progress is defined as the percentage of non-source nodes that were serviced.

should be around 800 rounds. In practice, however, the download was completed in about 750 rounds: this can be explained because, in the case of more file parts, there is a greater flexibility in choosing which part to take. However, this run-time advantage is somewhat mitigated by an increase in the size of the messages.

In order to compare the simulation results to a real content distribution network, we used experimental results collected by our team last year, using the planetary scale Planetlab testbed [6]. In this experiment, we used one source node, which had a full-size file of 130Mb divided into 130 parts of size of 1Mb. This experiment involved 200 clients spread over 100 worldwide sites, who simultaneously started to download the full file. The experiment's results are shown in Figure 7. The x-axis represents the number of finished machines (sorted)

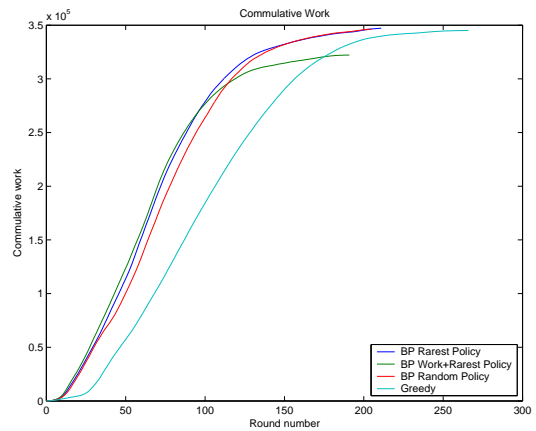


Fig. 5. Cumulative communication cost for the four policies. The combined policy of work+rarest part first achieves both lowest finishing time and lowest communication cost relative to the other policies.

and the y-axis is the completion time, in seconds. The two node-selection policies used were the uniform policy and the rarest-part first policy. Interestingly, the graph demonstrates that the curve behavior using 200 heterogeneous machines in the WAN is very similar to that of the simulation, even though the simulation used a simplified model which assumed synchronous rounds and homogeneous machines. Additionally, in the Planetlab experiment, the clients could connect to any other client, whereas our simulations used a fixed graph topology. In conclusion, despite the simplified model, we believe the simulation results are encouraging, since their similarity to the Planetlab results indicate that they still capture the essence of real life behavior of a planetary scale network.

VII. EXTENSION FOR ONLINE STREAMING SUPPORT

Online streaming audio and video pose a different challenge for content distribution networks, since their stream segments must arrive in order. This is because the audio/video players consume the audio/video stream while simultaneously downloading the next segments of the stream. Another requirement is the in-time delivery of the stream segments. In bulk data transfer, by contrast, there is no significance to the order at which the parts arrive, since the data is consumed only after the full file has been received at the destination. To support online streaming, we only need to add a *streaming policy*, which initializes the self potentials. The probability of taking a part decreases exponentially, based on the part's importance - its relative offset from the current stream offset. When consuming the real time stream, the part

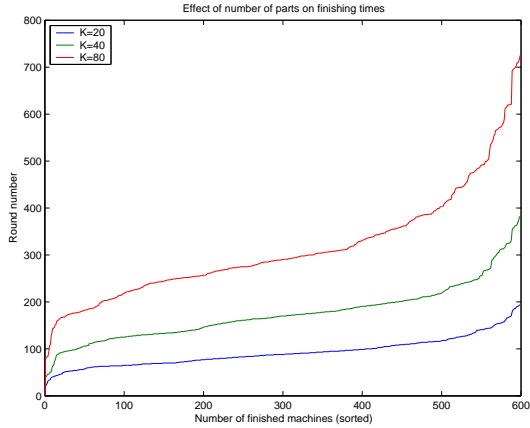


Fig. 6. Effect of file size on the system download performance.

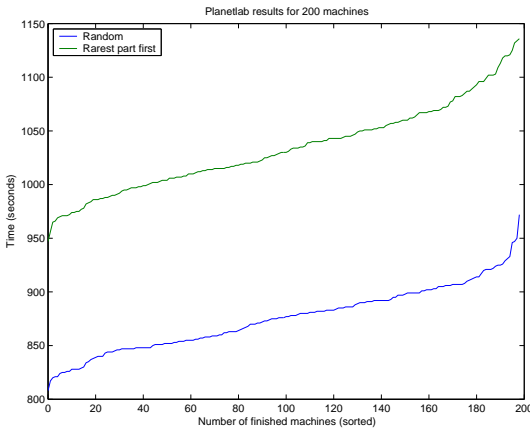


Fig. 7. Finishing times for 200 nodes on the Planetlab WAN testbed. Both policies used are greedy policies. The upper policy is uniform and the lower is rarest part first. We give this graph as a reference to show that the simulation curves are almost identical to the real life behavior of heterogeneous and unsynchronized machines.

that we need to play next is the most important. The next part is less important and so on. In general, the probability for taking a part with offset x is:

$$\psi_{ii}(F_x) = 2^{-(x-p)}$$

Where p is the current stream offset, $x > p$. And the cost function is the same as defined for the non-streaming case.

For example, assume a video player is now playing part 0, while it can take parts 1 to 4 from its neighbors. We assign a probability $1/2$ to the first part, $1/4$ to the second part, $1/8$ to the third and so on. Finally, we normalize these probabilities to get a probability distribution, which then becomes the self potential for node 0.

As we have shown above, no changes to the algorithm

are necessary in order to support online streaming. However, the efficiency measures for online streaming differ from those of bulk data. In the following paragraphs, we briefly outline the metrics used in the evaluation of our algorithm performance for streaming, and give some experimental results.

To evaluate streaming, we use several different metrics. The first is the Relative Delay Penalty (RDP), as defined in [5]; the RDP is the streaming delay we experience by using the overlay, relative to the optimal delay that can be achieved by IP multicast (optimal delay is calculated by taking the minimal spanning tree from the source). Two more important metrics are link load and node stress. Link load is the number of segments that traveled via a particular link³; Node stress is the number of forwarded segments, relative to the number of segments received, for each node. (In IP multicast, the maximal node stress is 1, since an intermediate node sends each segment once to all its neighbors). Lastly, we also consider the metrics of finishing time and of work, as we did for the content distribution problem.

To evaluate the streaming performance, we used a network of 100 nodes, with a stream of 100 segments. The topologies tested were transit stub and random. Each simulation was run 10 times and the results were averaged. Each simulation round began with one source node having the full stream, and finished after all nodes received the full stream. The latencies of the link were taken from the GT-ITM link weights. Only the GT-ITM results are presented here, since the random topology results were similar. The reference algorithms we used were application layer multicast tree and a minimal spanning tree.

Figure 8 compares the finishing times of the streaming policy, the combined work+rarest policy and the greedy algorithm. When using the streaming policy, nodes pay in their average download time - since we lose some of the flexibility in the part selection. Interestingly, the worst download time results remains constant using both of the policies.

Figure 10 shows the relative delay penalty when the streaming policy is used. The average RDP is about 20%; this means that nodes using the streaming policy experience (on average) a 20% delay receiving the segments, as compared to the optimal solution.

Link stress is shown in Figure 11. In the MST algorithm, the minimal spanning tree edges have the maximal link stress (100 parts), while the other edges

³This is a slightly different definition than the link stress that is used in [5]. We are using a different model where node links are physical links. Therefore, we are not building an overlay network.

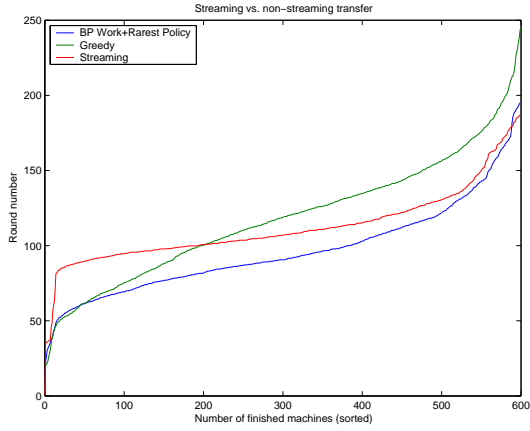


Fig. 8. Streaming policy vs. bulk-data policies

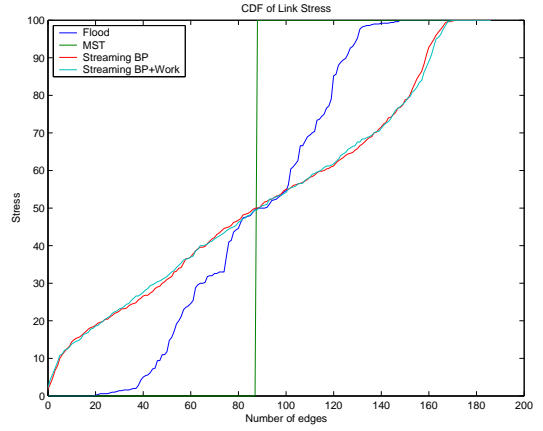


Fig. 11. Link stress of streaming policy vs. minimal spanning tree

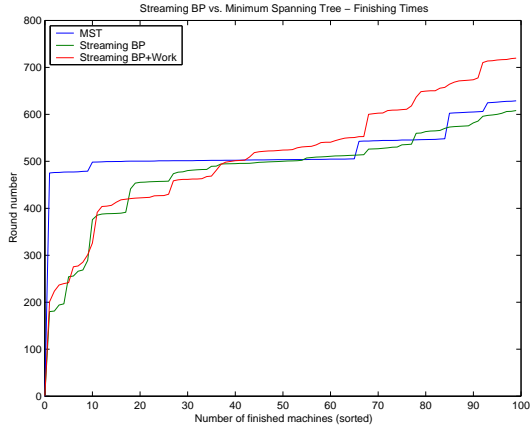


Fig. 9. Finishing times of 100 nodes using streaming policy vs. minimal spanning tree

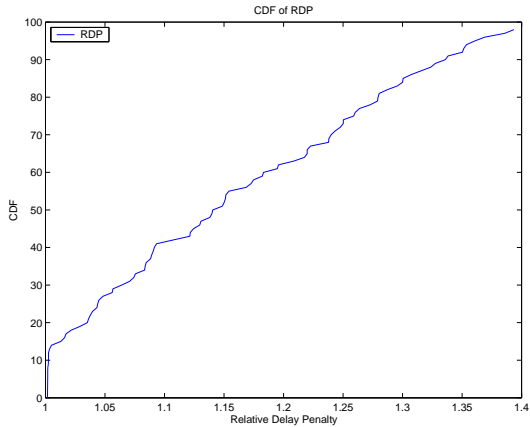


Fig. 10. Streaming policy Relative Delay Penalty (RDP)

have a link stress of zero. The flooding algorithm has better link stress since random edges are selected in each round for flooding the information. The streaming-BP algorithm has the best overall link stress, since the link stress distribution is much more uniform. The same results apply to the node stress, shown in Figure 12. Using the MST algorithm, nodes that are higher in the tree have higher node stress and the tree leaves have zero stress (since they are only receiving file parts and not forwarding them).

As expected, using the BP algorithm, the node stress is more uniformly distributed among the participating nodes.

To conclude, we have seen that the streaming policy works best in terms of running time and load balancing of nodes' work and network load. However, a disadvantage is the communication work, relative to the MST algorithm.

VIII. ALGORITHM PERFORMANCE UNDER FAILURES

Until now, we have assumed that nodes and communication links are reliable. However, we would also like to examine how our algorithm performs when faced with network failures. Initially, we choose a very simple failure model, where each node might fail i.i.d. with probability p in each round. We tested the BP algorithm under this failure model, and took a tree based topology as a reference. Intuitively, a failure of upper nodes in the tree might cause higher delay experiences in their descendants. The estimation, due to the fact that our solution is less structured, is that the failures will have a decreased effect on the download time. The simulation results are shown in Figure 13. The failure of 20% of the nodes resulted in an increase of less than 10% in the

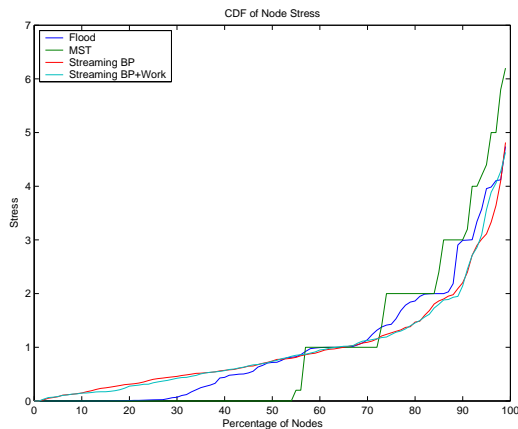


Fig. 12. Node stress of streaming policy vs. minimal spanning tree

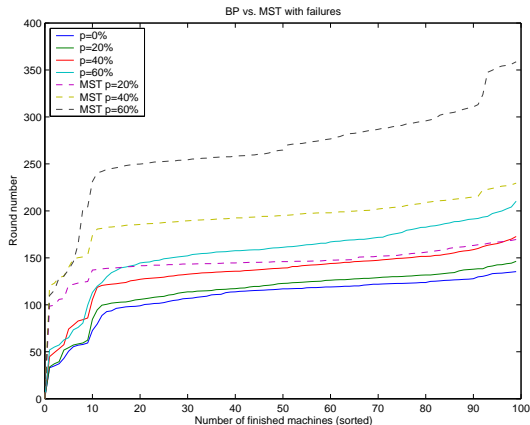


Fig. 13. BP algorithm vs. application layer multicast under node failures. Each node has a probability p to fail i.i.d. in each round.

total running time of the BP algorithm. Failure of 60% of the nodes resulted in an increase of only 20% in the BP running time. By contrast, with the application layer multicast tree, the running time increased linearly with the failure probability.

IX. RELATED WORK

Recently, another form of the content distribution problem was defined by Snoeren et. al. in [16], [17]. The problem was entitled the Fast Overlay Content Distribution (FOCD) and was proven to be NP-Complete. There are small differences between our models and the FOCD models: We assume the send/receive model, in which each node can send and receive one part on each round, whereas the FOCD assumes that each communication link can transfer several file parts in each round. Additionally, our problem assumes an initial state in which

one source node has the complete file and all the other nodes download the complete file. The formulation of the FOCD problem is more general: it allows for the algorithm to start from an arbitrary network state, and also supports that each node might download only a subset of the file parts. We believe that these differences should not affect a significant difference in the overall complexity of the problems - meaning that our problem is similar enough to FOCD to also be NP-Complete. Currently, however, we are not able to prove this.

There have been many proposed solutions to the content distribution problem. Tree or hierarchy based solutions are usually used for streaming [5], [25], [27]. Forests of trees are proposed in [21], [4]. Unstructured meshes provide enhanced fault tolerance [7], [18]. Traditionally, the solutions for streaming media are not used for bulk-data transfer, and vice versa.

Many of the proposed solutions for the content distribution problem use greedy or heuristic algorithms to decide which node to contact next or which part to take next. In BitTorrent [7], nodes select which part to take using the rarest-part policy. The download is performed from the set of neighbors with maximal available bandwidth. In Slurpie [24], nodes use a probabilistic backoff algorithm for balancing the load at the source node. In CoolStreaming [29] segments are selected, firstly, according to their rarity in the network, and secondly, by the node's available bandwidth. The drawback of all of the greedy solutions is that the nodes are not coordinated.

One interesting solution involves network coding [12] in which nodes are allowed to encode linear combinations of file parts and, in turn, achieve more flexible data flows. Decoding is performed after enough encoded parts are obtained. The drawback to this is that, normally, the decoding can be done only after a certain mass of file parts has been received - making the solution unsuitable for streaming.

X. CONCLUSION AND FUTURE WORK

In this paper, we considered the problem of large-scale content distribution and demonstrated that, through the use of a probabilistic graphical model, we can improve download performance in terms of time and communication cost, relative to the greedy solutions.

One possible area of improvement would be to replace the BP algorithm with gradient-descent algorithms that avoid the fluctuations towards a local optimum [26]. Another possible enhancement could be to create a list of future actions for a node, instead of deciding only on the next immediate action. Such a solution would reduce running time and with communication overhead costs, as well.

Another open question is how to derive an optimal algorithm for the content distribution problem.

REFERENCES

- [1] S. Banerjee, B. B., and C. Kommareddy. Scalable application layer multicast. In *Proceedings of SIGCOMM 2002 August, 2002*.
- [2] A. Bellissimo, P. Shenoy, and B. N. Levine. Exploring the use of bittorrent as the basis for a large trace repository. In *Technical report 04-41, June 2004*.
- [3] D. Bickson and D. Malkhi. The julia content distribution network. In *the 2nd Usenix of Real World Distributed Systems (WORLDS 05)*.
- [4] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *SOSP'03*, October 2003.
- [5] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proceedings of ACM SIGMETRICS, Santa Clara, CA, pp 1-12, June 2000*.
- [6] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.
- [7] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of P2P Economics Workshop, 2003*.
- [8] C. Crick and A. Pfeffer. Loopy belief propagation as a basis for communication networks. In *proceedings of the 19th Conference on Uncertainty in AI, 2003*.
- [9] K. C. Ellen W. Zegura and S. Bhattacharjee. How to model an internetwork. In *Proceedings of IEEE INFOCOM 1996, San Francisco, CA*.
- [10] F. L. Fessant, S. Handurukande, A. M. Kermarrec, and L. Massoulie. Clustering in peer-to-peer file sharing workloads. In *proc. of IPTPS 2004*.
- [11] P. Francis. Yoid: Extending the internet multicast architecture. In *Unpublished paper, April 2000*.
- [12] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *proc. of INFOCOM 2005, 2005*.
- [13] 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. In *proc. of ACM SOSP 2003*.
- [14] Ihler, Fisher, Moses, and Willsky. Nonparametric belief propagation for self-calibration in sensor networks. In *Information Processing in Sensor Networks 2004*.
- [15] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. A. Hamra, and L. Garces-Erice. "dissecting bittorrent: Five months in a torrent's lifetime". In *Passive and Active Measurements 2004, April 2004*.
- [16] C. Killian, M. Vrabie, A. C. Snoeren, A. Vahdat, and J. Pasquale. Brief announcement: The overlay network content distribution problem. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC) Las Vegas, NV, July 2005*.
- [17] C. Killian, M. Vrabie, A. C. Snoeren, A. Vahdat, and J. Pasquale. The overlay network content distribution problem. In *Technical Report CS2005-0824, UCSD, May 2005*.
- [18] Y. Kulbak and D. Bickson. The emule protocol specification. Technical report TR-2005-03, the Hebrew University of Jerusalem, 2005.
- [19] Massoulie and M. Vojnovic. Coupon replication systems. In *Sigmetrics 2005, June 2005*.
- [20] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *proc. of Uncertainty in AI, 1999*, pages 467–475.
- [21] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceesings Proc. of NOSSDAV, May 2002*.
- [22] M. Paskin and C. Guestrin. Robust probabilistic inference in distributed systems. In *the Twentieth Conference on Uncertainty in Artificial Intelligence (UAI 2004), Banff, Canada, July 2004*.
- [23] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *4th International Workshop on Peer-to-Peer Systems (IPTPS'05), Feb 2005*.
- [24] R. Sherwood, R. Braud, and B. Bhattacharjee. Slurpie: A cooperative bulk data transfer protocol. In *proc. of IEEE Infocom 04', March 2004*.
- [25] D. A. Tran, K. A. Hua, and T. T. Do. A peer-to-peer architecture for media streaming. 2003.
- [26] M. J. Wainwright, T. Jaakkola, and A. S. Willsky. Exact map estimates by (hyper)tree agreement. In *Advances in Neural Processing Systems (NIPS), MIT Press, 2003*.
- [27] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *Proceedings of Fourth International Workshop on Networked Group Communication (NGC), October 2002*.
- [28] Y. Weiss and W. T. Freeman. On the optimiality of solutions of the max-product belief propagation algorithm in arbitrary graphs. In *IEEE Transactions on Information Theory 47:2 pages 723-735, 2001*.
- [29] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. Donet/coolstreaming: A data-driven overlay network for live media streaming. In *IEEE INFOCOM'05, Miami, FL, USA, March 2005*.