The Overlay Network Content Distribution Problem

Chip Killian, Michael Vrable, Alex C. Snoeren, Amin Vahdat, and Joseph Pasquale
University of California, San Diego
{ckillian,mvrable,snoeren,vahdat,pasquale}@cs.ucsd.edu

ABSTRACT

Due to the lack of deployment of a network-layer multicast service, many overlay multicast protocols have been designed and deployed across the Internet to support content distribution. To our knowledge, however, none have provided a rigorous analysis of the problem or the effectiveness of their proposed solutions. Here, we set aside the engineering challenges of protocol design to focus on the fundamental graph problem.

We begin by formulating the Overlay Network Content Distribution (OCD) problem and show that variants that attempt to optimize for either speed or bandwidth utilization are NP-complete. Using both a time-indexed Integer Program and a branch-and-bound search strategy, we calculate optimal solutions for small graphs. While solutions to OCD provide performance bounds, realistic deployment scenarios will not have global network information. Hence, we introduce an on-line variant, the Local-knowledge Overlay Content Distribution (LOCD) problem and show that no constant-competitive approximation exists. Instead, we present several heuristics that perform well in realistic topologies. We conclude with an evaluation of our global and local heuristics and enumerate a number of open problems.

1. INTRODUCTION

Recently, there has been tremendous interest in building cooperative content distribution networks to deliver data among a group of nodes spread across a wide-area network. There are multiple degrees of freedom in defining the problem, including: i) the number of senders, ii) the subset of data each receiver is interested in, and iii) whether the cooperation is meant to be one-shot or long-lasting. Similarly, there are a variety of goals for such systems. The most common goal is to improve performance, i.e., to remove the bottleneck associated with a single source attempting to disseminate content to a large number of receivers. Other target metrics include fairness (ensuring that nodes contribute roughly in proportion to one another), reliability, and per-object latency. Even for the baseline goal of maximizing performance, there are multiple variants including minimizing average performance, worst-case performance, and the total global bandwidth required to achieve a performance objective.

Given the very large space of reasonable problem formulations, combined with the peculiarities of Internet graph structures, varying bandwidth and latency characteristics, failures, and TCP congestion control behavior, it is not surprising that there have been a significant number of distinct proposed system architectures, each with a particular variation on the problem statement and with a particular set of benefits [2, 3, 4, 5, 9, 11, 12, 16, 18, 19].

Perhaps surprisingly, however, there has not been a precise formulation of the underlying problem that is being addressed and, hence, no understanding of how a particular solution compares to calculated upper/lower bounds (depending on the exact formulation) for the problem, either exact or approximated based on available heuristics. For example, in our own work on comparing the performance of a variety of content distribution systems under a variety of emulated and deployed network settings [10], we had difficulty arguing how well we were doing relative to how well *any* system could perform.

Thus, the goal of this paper is to formally define the *Overlay Network Content Distribution* (OCD) problem and show how our problem formulation can be generalized to a wide range of existing and potential scenarios being actively pursued by the research and development community. We also show that the problem is NP-complete and, that while it is similar to a number of well-known problems in the theory community, details of the deployment scenario make it distinct from the problems of which we are aware. In particular, nodes may duplicate individual items along multiple output links and the distribution structure is not necessarily restricted to a tree but may take on an arbitrary mesh of interconnectivity over an underlying graph.

As a starting point for understanding the behavior of this problem, we develop a simple algorithm and an integer programming formulation to calculate optimal behavior for small graphs with few files. We also develop a global heuristic to provide approximate answers for larger problem settings and on-line heuristics that are more amenable to translation to deployment in realistic large-scale network environments. Finally, we calculate bounds (not necessarily tight) to provide a rough notion of the quality of our local and global heuristics in a range of scenarios.

We hope that our formulation of the Overlay Network Content Distribution problem will lead to subsequent approximation algorithms that can, for instance, guarantee approximations with bounded error in realist settings, perhaps by mapping the problem to other known settings. Further, we believe that our findings regarding the behavior of a variety of heuristics have potential implications for how next-generation overlay content distribution networks should be architected and deployed as well as whether significant performance improvements are still available relative to a given implementation and system architecture.

The rest of this paper is organized as follows. We begin in Section 2 with a survey of related work. Section 3 formally defines the problem, as well as providing hardness results. We consider the on-line formulation in Section 4. Section 5 details the performance results of our heuristics. We conclude in Section 6 with a brief enumeration of open problems in this space.

2. RELATED WORK

Overlay content distribution has received a great deal of attention over the past few years. In general, overlay networks construct a single distribution topology—traditionally a spanning tree, but more recently meshes have come into favor—which they then use to forward traffic from source to destination. Systems have focused, with varying degrees, on constructing topologies that reduce bandwidth consumption, distribution latency, or data loss. We are not considering lossy channels or node failures here, so we will restrict our survey to systems that focus on the first two.

Overcast [9], one of the first overlay systems proposed, attempts to construct a bandwidth-optimized overlay tree. An incoming node joins at the source and probes for acceptable bandwidth under one of its siblings to descend down the tree. Obviously, a node's bandwidth and reliability is determined by characteristics of the network between itself and its parent.

In an attempt to improve distribution speed, Narada [5] first constructs a bandwidth and latency-sensitive mesh between all nodes based on a *k-spanner* graph. Using the inter-node characteristics gathered from the mesh, Narada selects per-source spanning trees for forwarding data. Snoeren *et al.* [18] ignore bandwidth costs, and focus on reliable low-latency delivery by constructing a mesh consisting of k edge and node disjoint spanning trees. Young *et al.* [19] construct an overlay mesh of k edge-disjoint minimum cost spanning trees (MSTs). The algorithm for distributed construction of trees uses overlay link metric information such as latency, loss rate, or bandwidth.

SplitStream [3] aims to construct an interior-node disjoint forest of k Scribe [15] trees on top of a scalable peer-to-peer substrate [14]. The content is split into k *stripes*, each of which is pushed along one of the trees. This system accounts for physical inbound and outbound link bandwidth of a node to determine the number of stripes a node can forward. In CoopNet [12], the source of the multimedia content computes locally random or node-disjoint forests of trees in a manner similar to SplitStream. The trees are primarily designed for resilience to node departures, with network efficiency as the second goal. Similarly, the FastReplica [4] file distribution system alleviates the overlay tree's problem of reliance on a single node for high bandwidth dissemination. The source of a file divides the file into n blocks, sends a different block to each of the receivers, and then instructs the receivers to retrieve the blocks from each other.

A popular alternative to structured content distribution overlays, BitTorrent [2] uses peer-to-peer communication to distribute large files on the Internet. Incoming nodes rely on a centralized *tracker* to provide them with a list of existing system participants and system-wide block distribution for random peering. No attempt is made, however to provide optimal bandwidth or latency guarantees. Slurpie [16] improves upon the performance of BitTorrent by using an adaptive downloading mechanism to scale the number of peers a node should have.

Kostić *et al.* recently proposed an improved version of Bullet [10, 11] that outperforms previous overlay systems in terms of download time. In doing so, they consider a number of engineering tradeoffs common to overlay systems. Somewhat surprisingly, however, neither they nor anyone else that we are aware of have rigorously formulated or analyzed the underlying graph problem. The file distribution problem is related, however, to several classical problems in operations research.

Perhaps the most fundamental is the well-known transportation problem, in which goods are manufactured in particular locations, and need to be delivered to consumers in other locations. Given the cost of transportation between any two cities, the challenge is to find an assignment of suppliers to consumers of minimal cost that satisfies demand.

The transportation problem is a specific instance of a class of problems known as network flow. Network flow considers a weighted, directed graph with a source and a sink, and computes the maximum flow rate from source to sink subject to edge capacities (weights). Many generalizations exist, including multi-commodity flows, and generalized network flow, where edges are allowed to have a gain: the rate of the flow leaving an edge is a linear factor of the flow entering the edge. The defining constraint of a network flow problem, however, is the notion of flow conservation: the sum of the flow leaving a node must be exactly equal to the sum of the flow entering the node. This is clearly not the case in our situation, as incoming data can be both stored[1] and duplicated.

The overlay content distribution problem is further complicated by the fact that data is not quite like water: each data element is distinct, and a node cannot forward a particular piece of data until it has been received. Similar precedence relations have been considered in the context of the classical machine scheduling problem, where the task is to complete a given set of jobs in the shortest amount of time, subject to release constraints. The problem has been extended to network scheduling, where each job originates at some network node but can be processed at other nodes in the network. Moving a job between machines incurs some latency however, and the goal is to reduce total completion time [13]. Unfortunately, there is no notion of job duplication, or a particular affinity of jobs to machines.

The family of work most closely related to ours is likely the study of online routing [1] and admission control, but these problems assume a circuit-like construct, where an edge can be assigned some number of circuits at a time, and the task is to schedule given calls as efficiently as possible. Again, there is no notion of storage or duplication. Recent work by Goel *et al.* studies the so-called online FTP problem [7], which is an instance of the more general set scheduling problem. Unfortunately, they too consider flows as single-source, single-receiver, and do not deal with the notion of replication.

[1] We note that storage is not hard to model in the traditional network flow sense: simply add self-edges of infinite capacity at each node.

3. PROBLEM DESCRIPTION

We begin by describing the generalized system model we will use to formulate the overlay content distribution problem and its variants. We assume, without loss of generality, that all content is in the form of unit-sized *tokens*; files can be represented as sets of tokens. Tokens start out at one or more nodes (senders), and the goal is to transfer them to a different set of nodes (receivers). For the purposes of this paper, we will assume all nodes in the network are overlay participants; hence, they can store, forward, and duplicate tokens at will. We also assume the network is private: link capacities are constant, no tokens are lost, and latency does not change with load—any number of tokens, up to the capacity of the link, can be transferred across a link in unit time.

3.1 General model

The input is a simple, weighted directed graph $G = (V, E)$, a set of tokens T, and two functions $h : V \rightarrow 2^T$ and $w : V \rightarrow 2^T$. Here, 2^T denotes the power set of T. Let $c : E \rightarrow \mathbb{N}$ denote the function which gives the weight (capacity) of each arc. (Note multi-arcs can be represented as a single arc whose capacity is the sum of the multi-arcs.) The h (have) function denotes the set of tokens that each vertex in the graph initially possesses. The w (want) function indicates which tokens each vertex would like to eventually possess.

We will define a move as an assignment of a token to an arc, and a *timestep* to be a set of simultaneous moves. A *distribution schedule* (or simply *schedule*) proceeds as a sequence of timesteps. At each timestep, the number of tokens that may be assigned to an arc is limited by its capacity $c(u, v)$, and a token may only be sent by a vertex if that vertex possesses a copy of the token at the start of the timestep.

More formally, let $t \in \mathbb{N}$ denote the length of a schedule (number of timesteps). The schedule is defined by a collection of functions $s_i : E \rightarrow 2^T$, where $0 \leq i < t$; the function s_i gives the set of tokens that are sent across arc (u, v) during timestep i. For a schedule to be valid, we must be able to construct a set of functions $p_i : V \rightarrow 2^T$ for $0 \leq i \leq t$ that specify which tokens a vertex possesses at each timestep. The schedule is subject to the following restrictions:

$$p_i(v) = \bigcup_{\substack{u \in V: \\ (u,v) \in E}} p_{i-1}(v) \cup s_{i-1}(u, v)$$

$$\begin{array}{rcll} p_0(v) & = & h(v) & \text{(Initial assignment)} \\ |s_i(u, v)| & \leq & c(u, v) & \text{(Capacity)} \\ s_i(u, v) & \subseteq & p_i(u) & \text{(Possession)} \end{array}$$

A distribution schedule is said to be *successful* if $w(v) \subseteq p_t(v)$ for all vertices $v \in V$. There are two obvious interesting characteristics of successful schedules: how many timesteps they contain, and their total bandwidth consumption (how many tokens are transferred across an arc; this is equivalent to the number of moves).

3.2 File distribution times

Define the *Fast Overlay Content Distribution* (FOCD) problem as determining a satisfying distribution schedule of minimum length, τ. The problem is satisfiable if there exists some τ for which this is true. The *Decisional Fast Overlay Content Distribution* (DFOCD) problem takes as input a Fast Overlay Content Distribution problem and an integer τ^*, and determines whether the Overlay Content Distribution problem is satisfiable in τ^* steps. Here, τ can be thought of as the maximum completion time over all downloads. This metric is often referred to as *makespan* in the scheduling literature.

Before proceeding to other metrics, we briefly consider the difficulty of FOCD. For the purposes of discussing problem sizes, let $n = |V|$ and $m = |T|$.

THEOREM 1. *If an instance of FOCD is satisfiable, it is satisfiable in* $m(n-1)$ *moves.*

PROOF. Suppose we are given some successful run for an FOCD instance. No useful work is accomplished if a vertex receives a copy of a token it already possesses, so any such moves can be removed. There are n vertices and m tokens, each of which is initially possessed by at least one vertex, so there are only $m(n-1)$ possible moves that might be made after our above cleanup. □

Corollary: Any satisfiable instance of FOCD may be satisfied in $m(n-1)$ steps (in the worst case, only a single move is made at each timestep).

THEOREM 2. *If an instance of FOCD is satisfiable, then there exists a successful run that can be described in* $O(nm \cdot (\log n + \log m))$ *bits.*

PROOF. This follows from the previous theorem. As remarked, there is a schedule consisting of at most $m(n-1)$ moves. Each move can be described by listing a token and the arc to move it across. This information can be encoded with $2\log n + \log m$ bits (there are at most $n(n-1)$ arcs). The sequence of moves can be segmented into timesteps by giving a list of the number of moves that make up each timestep, in $\log(nm)$ bits since there are no more than $m(n-1)$ moves per timestep. In total, the description takes $O(nm(\log n + \log m))$ bits as claimed. □

THEOREM 3. *FOCD is NP-complete.*

PROOF. We need to show both that FOCD is NP-hard, and that a solution to FOCD can be verified in polynomial time. For any instance (G, T, h, v) of FOCD, we saw above that the size of a solution to the FOCD problem consists of no more than $O(nm(\log n + \log m))$, which is polynomial in n and m. It is easy to see that validity of each move (that the token is present at the source) can be tested in polynomial time, as can arc capacity and the starting and ending conditions. In the appendix, we show FOCD is NP-hard by constructing a reduction for the Dominating Set problem, which is known to be NP-complete. □

3.3 Bandwidth constraints

Rather than focusing entirely on speed, we can instead consider the minimum bandwidth necessary to distribute the tokens to the sinks. If we consider sending one token along one edge to take one unit of bandwidth, then finding the minimum-bandwidth distribution schedule is only a small modification: that is, find a distribution schedule subject to the same constraints as before, but try to minimize the number of moves

$$\sum_{\substack{0 \leq i < t \\ (u,v) \in E}} |s_i(u, v)|$$

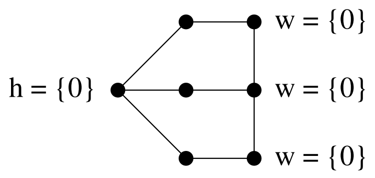


Figure 1: A graph in which minimizing the time taken and the bandwidth required are at odds. The minimum time schedule takes 2 timesteps and uses 6 units of bandwidth; a minimum bandwidth schedule uses 4 units of bandwidth but takes 3 timesteps.

rather than the number of timesteps t. We call this problem the *Efficient Overlay Content Distribution* (EOCD) problem. Note that simultaneously minimizing the number of timesteps and the bandwidth used is not always possible; the graph in Figure 1 illustrates such a case.

It is straightforward to see that EOCD reduces to the generalized Steiner tree problem, which is known to be NP-complete [8]. To distribute any token using the minimum bandwidth is to distribute it along the min-cost tree from its source(s) to all nodes that want that token with unit-cost edges. If we do not care about number of timesteps, then optimal bandwidth can be achieved by distributing each token serially over the Steiner tree to the nodes that want it. (It is a small adaptation to the traditional Steiner tree problem to handle the case where a token is initially at multiple vertexes: augment the graph with 0-cost arcs between all vertices that initially have the token). Therefore it is a series of Steiner tree problems that, in the worst case, takes 2^{nm} timesteps.

3.4 Integer programming formulation

We now construct a time-indexed integer program that attempts to provide a solution to EOCD. First, extend graph G by adding a self-arc at every vertex. That is, $E' = E \cup \bigcup_{v \in V}(v, v)$. Create a variable $x^i_{(u,v)t}$ for each arc (u, v) and token t at timestep i. Let $w_{vt} = 1$ if $t \in w(v)$, 0 otherwise. We are now interested in the solution to:

$$\text{Min} \quad \sum_{\substack{0<i\le\tau \\ (u,v)\in E, t\in T}} x^i_{(u,v)t}$$

subject to:

$$x^i_{(u,v)t} \le \sum_{\substack{v\in V: \\ (u,v)\in E'}} x^{i-1}_{(u,v)t}$$

$$\sum_{t\in T} x^i_{(u,v)t} \le c(u,v)\ \forall i \le \tau, \forall (u,v) \in E$$

$$x^{\tau+1}_{(v,v)t} \ge w_{vt}$$

$$x^i_{(u,v)t} \in \{0,1\}$$

where the initial conditions are given by $x^0_{(v,v)t} = 1$ if $t \in h(v)$, 0 otherwise. It is easy to see that any solution to the IP is a solution to EOCD. For any solution to IP, define

$$p_i(v) = \bigcup_{t:x^{i+1}_{(v,v)t}=1} t, \quad s_i(u,v) = \bigcup_{t:x^i_{(u,v)t}=1} t.$$

In realistic situations, we will not be interested solely in minimizing time or bandwidth, but rather to find some balance. One such approach is to search for a bandwidth-optimal solution subject to the constraint that the time be no more than some constant factor of the optimal time, or vice versa (time-optimal subject to a bandwidth constraint). Formalizing this hybrid goal is a subject of ongoing work.

4. ON-LINE APPROXIMATION

Our file distribution model is not an entirely realistic model for how a file might be distributed. In many cases, we cannot expect to know the entire state of the system at the start, so as to construct a global plan for which tokens should be sent where. A more realistic model is to assume that each node makes a decision about which tokens to send at each timestep based only on knowledge it has acquired from its neighbors.

Hence, we now consider an on-line algorithm that attempts to solve the more realistic problem. In this case, information about which tokens are available and requested is all available at the start of the computation, though not known by every node. We ask for an algorithm that can run at each node, making only local decisions that will eventually ensure that all tokens are distributed where needed. We also care about how close such an algorithm can come to achieving the optimum distribution time.

4.1 Local knowledge

We formulate the *Local-knowledge Overlay Content Distribution* (LOCD) problem to capture this idea. The basic problem is as before, but we formalize the notion of information that can be known by each vertex at any timestep, and require that any algorithm only make decisions based on this local information.

Let $k_i(v)$ denote the *knowledge* of vertex v at the start of timestep i. We require that $k_0(v)$ be computed by a deterministic function of the list of neighbors of vertex v, the capacity of each edge incident upon v, $h(v)$, and $w(v)$. (Optionally, we might expand the initial knowledge of a vertex to include additional information about the graph topology, or other similar information.)

The decision of which tokens should be sent along which arcs out of a vertex v must be made entirely using current knowledge of a vertex. That is, at timestep i, for any edge (v_1, v_2), $s_i(v_1, v_2)$ must a function only of $k_i(v)$. If we wish to allow randomized algorithms, s_i may be a function of $k_i(v)$ and the output of a random number generator, but no other information.

The knowledge of each vertex may change at each timestep as it learns information from its neighbors. We require that $k_{i+1}(v)$ be computable by a deterministic function that takes as input only:

- Previous knowledge: $k_i(v)$.
- Knowledge acquired from neighbors: $k_i(u)$ where either $(u, v) \in E$ or $(v, u) \in E$. We allow information to travel bidirectionally along an edge, since even if an edge is only unidirectional, it may be useful to send "want" information back to the sender.

- If a randomized algorithm is used, the behavior of node v and its neighbors at the previous timestep. (For deterministic algorithms, this is not needed as the behavior of each vertex on the previous timestep can be computed from knowledge received from it.)

Questions that may be asked about the local-decision problem include: For a given local-decision algorithm, will all tokens eventually be distributed to vertices that want them? If so, is there a bound that can be given for the time taken, relative to the optimal solution (with global knowledge)? Can any lower bounds for the additional time required be made that are independent of the algorithm chosen?

4.2 Non-optimality in the local case

The performance of on-line algorithms is typically modeled using competitive analysis [17], where the performance of the on-line variant is compared to that of an optimal prescient algorithm on any sequence of events (in our case, set of tokens T, sources $h(v)$, and receivers $w(v)$).

We observe that it will not, in general, be possible for a local-decision algorithm to perform as well as the optimal solution to FOCD. It is possible for an on-line algorithm to always perform within an additive factor of the diameter of the graph, however, since with this many steps at the start of computation, full information about the state of the graph can be propagated to each vertex. Armed with this knowledge, each vertex can compute an optimal solution for the entire graph (deterministically), then follow this schedule to distribute the tokens. We are hopeful more sophisticated approximations may exist, but it is clear their efficiency must depend on the characteristics of the graph.

THEOREM 4. *There exists no c-competitive on-line algorithm for FOCD for any fixed constant c.*

Due to space constraints, we provide only a brief sketch of the proof. Consider the situation of two maximally-separated vertices in which one has tokens that the other requires. If the sender has many tokens that the receiver does not want, then simply sending out tokens in the hopes they are useful cannot speed up the solution beyond waiting to hear knowledge of which tokens are needed. A similar argument can be made for the EOCD problem, but the bound depends on the bandwidth cost of sending knowledge.

5. EVALUATION

Given the complexity of the offline problem and the lack of provably competitive online approximation algorithms, we instead consider the empirical performance of several heuristics. We design both global (offline) and local (online) heuristics, and simulate their performance over a number of interesting cases.

5.1 Heuristics

We begin by considering several straight-forward heuristics for the online problem.

Round Robin The round-robin strategy simply sends the circular queue of tokens over each link (skipping tokens it does not have). This is the simplest of the heuristics, and can easily be computed locally as no information other than the set of tokens kept locally and the last token sent to each peer. While simple, this strategy suffers from sending tokens multiple times to peers and of duplicating sends that other peers have also sent.

Random We next move on to a basic random heuristic. In this heuristic we assume that peers have current knowledge about the tokens known by each of their peers at the beginning of the turn. Each vertex then independently chooses at random which tokens to send over the edge. How a vertex would know this information is an implementation problem, but it is reasonable to believe that peers can communicate this information at the granularity of a turn with good accuracy. Further exploration may also relax this requirement, instead allowing peers to know about the state 'k' turns ago of their peers.

Local The design of our local heuristic is based on the commonly proposed notion of "rarest random." Rarest random is often used in multicast flooding because, by diversifying the set of tokens known by various vertices, they can share them with each other for increased bandwidth. To effect rarest random, we therefore have to make some assumption about a vertex's knowledge of rarity. For simplicity, we have assumed that at every time step, the step's initial aggregate need and knowledge are distributed to all vertices. This could be implemented by a multicast tree, but the details of such are ignored at present, though we recognize the potential need to support a delay in the aggregate knowledge known. To avoid the problem where two peers send the same "rare" block in the same direction, our heuristic subdivides a vertex's needs to their peers. This is analogous to a request for blocks. A final note about rarest random is that prior work typically considers all receivers wanting all files, so only one aggregate vector is needed. To handle the general problem, we distribute both aggregates of what vertices want and what they do not have.

Bandwidth One problem with the aforementioned techniques is that they are constructed with no concern for the bandwith they consume, and are always content to flood data across any link where it can increase knowledge. As a result, we developed an online heuristic, albeit with global knowledge, which more cautiously adds tokens to a move. This bandwidth heuristic is designed on the principle that each vertex shall obtain from its peers in its next turn only tokens that it will eventually use. We then determine whether a vertex will use the token by i) if it needs the token, or ii) if it is the closest one-hop-knowledge vertex to a node that needs it. A one-hop-knowledge vertex is one which for a given token, *could* obtain the token in a single turn given the opportunity.

Turning to global techniques, we consider the general case of the local heuristic described above.

Global In addition to the aggregate vector, vertices have the ability to coordinate across each other at each timestep to ensure that they maximize diversity. This also alleviates the need for vertices to request tokens from other vertices since there is global coordination. Our implementation of this technique applies a greedy selection algorithm over the set of tokens and edges, and is thus not guaranteed to maximize diversity. This decision was made to allow the heuristic to function at large scale, and we are currently considering ways to improve our greedy algorithm.

In the offline case, regardless of the algorithm used to compute a schedule, once a satisfing schedule is found, we can go back and prune any unnecessary moves, reducing the bandwidth consumption. Pruning first removes all moves that deliver a token repeatedly to the same vertex, and then works back from the last move to the first, removing moves that deliver tokens which were never used by the destination vertex.

Finally, in an effort to efficiently compute performance bounds for our algorithms on large graphs, we develop lower bound approximations for both remaining timesteps and remaining bandwidth. The remaining bandwidth algorithm we use is very simple, counting every token that is wanted but not known at each vertex. Logically this represents the bandwidth that would be consumed if the schedule could be completed in a single timestep. The remaining move count algorithm is a bit more complicated. We define $M_i(v)$ as the number of moves vertex v would need to retrieve all tokens if all tokens within a radius of i could be retrieved in i timesteps. This boils down to $i + |T^{c_i(v)}|/indegree$, where $|T^{c_i(v)}|$ is the number of tokens outside the closure around v of a radius of i. We then take the maximum of all values of i. While not a tight bound, this approximation can account both for tokens that are too far away to be retrieved quickly and tokens that cannot be retrieved fast enough due to a limited incoming capacity. Additionally, we also consider as a special case looking ahead one timestep, since we can immediately compute how many tokens can be retrieved in a single timestep.

5.2 Single file

We begin our evaluation by applying our heuristics to single-file content dissemination, considering how graph size and receiver density affect distribution. For the single-file case we consider both random graphs and transit-stub graphs generated by the GT-ITM topology generator. For the single file results, we are considering a file of 200 tokens, and edge weights chosen randomly between 3 and 15 tokens. These assignments are arbitrary, but chosen to capture the variety of real vertex connectedness.

Graph size. First we consider the number of moves and bandwidth used when a single source distributes a file to all vertices. In this case we run with graphs from 20 to 1000 vertices, randomly adding edges with uniform probability $2 \ln n/n$. At this probability, the number of edges in the graph grows as $O(n \ln n)$, which maintains reasonable connectedness. We generate several instances of the graph for each size graph, and repeat our heuristics 3 times[2] for each graph.

Figures 2 and 3 show that the number of moves needed does not correlate with the number of vertices. The number of moves taken for any particular graph seems to be more dependent upon the actual connectivity of the graphs and, in particular, their random edge weights. As mentioned, we have chosen the edge connection threshold to keep the connectivity at a level to support even distribution as graph size grows. On the other hand, the bandwidth consumed grows roughly linearly with the number of vertices of the graph. We also clearly see that the round robin technique, though successful, is much slower than the other techniques that account in one way or another for what their peers have, and that the bandwidth heuristic is slower than the others when all nodes want everything, and also does not demonstrate any savings when compared with random. Furthermore, from the perspective of the number of moves and bandwidth, random performs within a constant factor of the smarter heuristics. This case of all vertices wanting all files is unique however, since no bandwidth is wasted as long as tokens aren't re-sent.

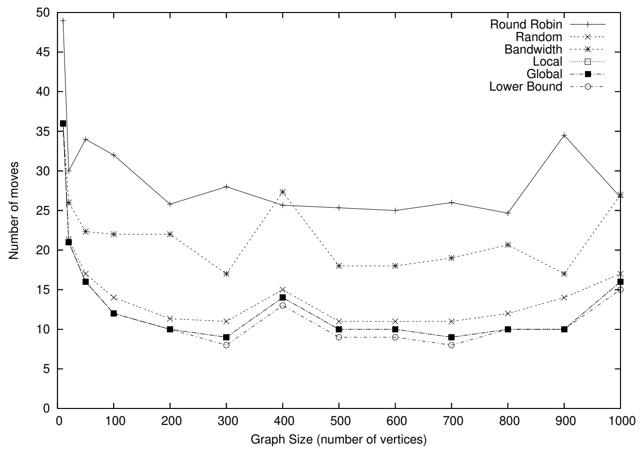
Receiver density. The single-source, all receiver problem is the most common one from the systems currently being built. Its inclusion demonstrates how though flooding is a robust solution to disseminate content, due to the circumstances it is exempt from bandwidth waste. But we expect that when flooding to a set of peers who do not want the file it will use far more bandwidth than necessary. We measured this using a single source, single file distribution over 200 nodes that were added to the want set depending on their randomly generated score. We performed the same experiment with both random and transit-stub graphs, but have only included random since as before it is representative of both.

Figure 4 shows the result of this experiment both in terms of bandwidth and number of moves. On the x-axis is the score threshold that we used to add the vertices to want set with. What we see from these resuls is that random continues to do well with respect to number of moves, but is now roughly double the other heuristics in bandwidth consumed. Also we note that the bandwidth consumed and the number of moves is roughly constant for the flooding heuristics so they are not taking advanage of the smaller number of vertices that want the tokens.. By constrast, the bandwidth optimizing heuristic, which is slower than the others by a small percentage, takes much less bandwidth than all heuristics when the threshold is small, and continues to use less bandwidth than random until the threshold returns to 1. Finally, the pruned bandwidth of the heuristics is roughly optimal, and demonstrates how the flooding runs nearly the same regardless of how many nodes want the file.

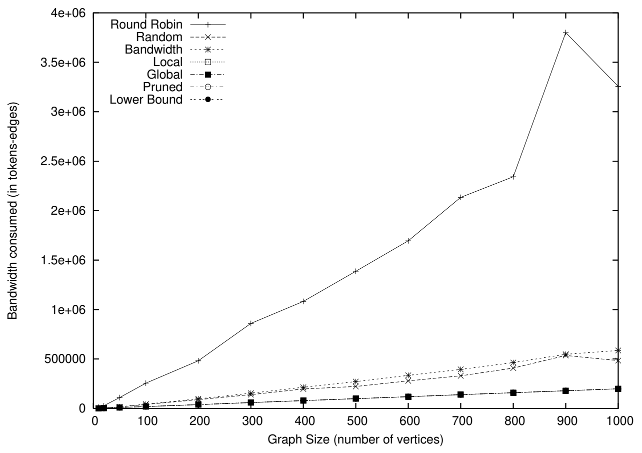
5.3 Multiple files

Number of files. The multi-file scenario was developed by starting with 200 vertices and 512 tokens at a single source. Initially, all vertices wanted all 512 tokens (1 file). Then, we subdivide both the file and the vertices, and then each set of 100 nodes wanted 1 of the 2 files, containing 256 tokens each. This subdivision process is repeated until there are 128 files of 4 tokens each which are wanted by 1 or 2 vertices each. What remains constant across this graph is the number of tokens that need to be distributed from the single source. As before, the edge weights are chosen at random from 3 to 15. Given our earlier observation about transit-stub graphs, we continue to present results from random graphs only.

[2]The variation of the interesting heuristics is very small, typically at most 1 move.

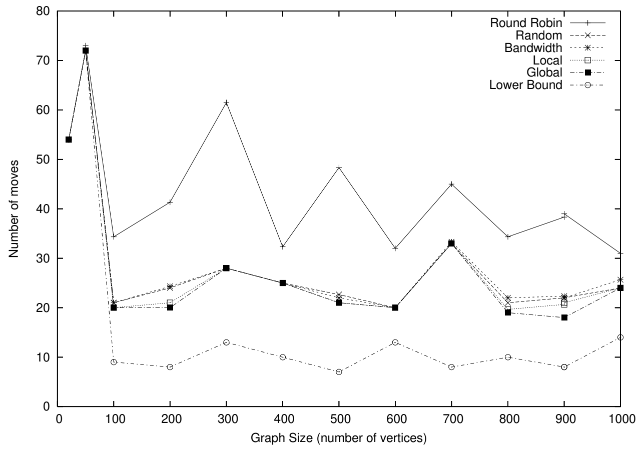


(a) Moves

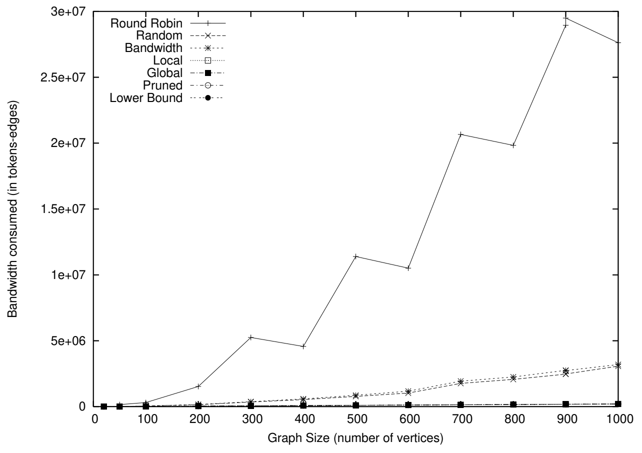


(b) Bandwidth

Figure 2: Moves and Bandwidth as a function of graph size. Single source and file to all receivers on a random GT-ITM graph.



(a) Moves



(b) Bandwidth

Figure 3: Moves and Bandwidth as a function of graph size. Single source and file to all receivers on a transit-stub GT-ITM graph.

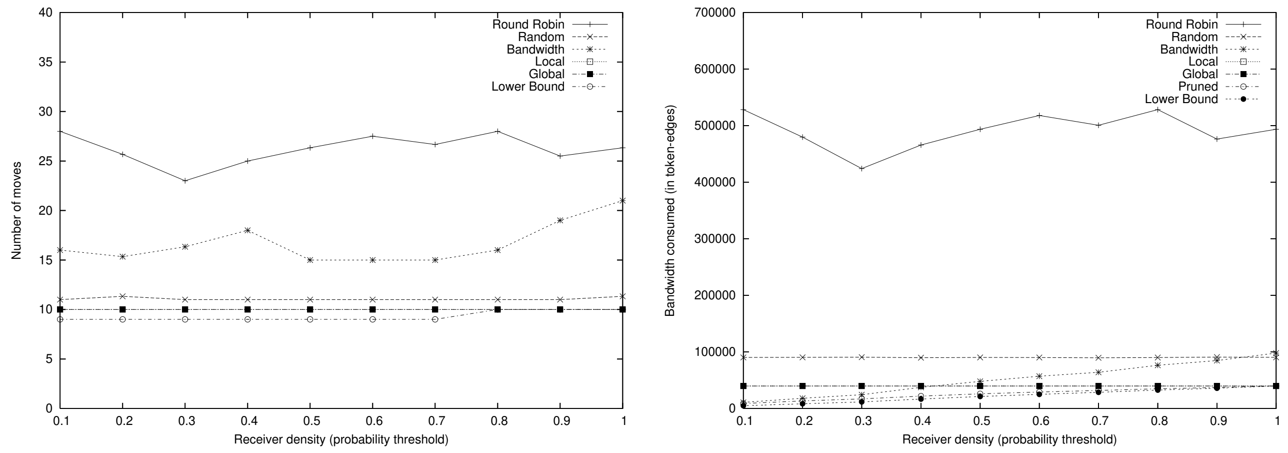
In Figure 5, we see that after an initial large descent in number of moves due to the number of tokens being sent to a bottleneck vertex, nearly all heuristics level off. Given that their bandwith consumption is also roughly even, this suggests that once again they are performing the same distribution regardless of how the files are broken up (i.e., sending all tokens to all nodes). Furthermore, random continues to perform within a constant factor of the other flooding heuristics, though its separation is now more pronounced. Here, only the bandwidth heuristic varies and improves as the files need to go in more constrained directions, though even at its best it still takes as many turns on this graph as the flooding protocols. However, bandwidth is substantially lower, tracking both the lower bound as well as the pruned versions of the flooding heuristics.

Multiple senders. The multiple-sender is an adaptation of the number of files scenario above where the source of each file was randomly chosen from the set of vertices which did not want it. Figure 6 closely mimics the Figure 5, so we can observe the same trends whether the files begins at a single place or multiple places. This also demonstrates that although in the distributed file case the number of tokens per source vertex have decreased the same trend in heuristics occurs.

6. CONCLUSION

In this paper, we formalize the overlay network content distribution problem, a problem with applicability to a wide variety of recent efforts. Currently, efforts into building faster, more reliable, lower latency, etc. overlays are limited by a lack of understanding of the bounds available in a given scenario for a given problem formulation. We show that the problem is NP-complete and present a number of global and online heuristics for a number of problem variants and network topologies. Of course, there are a number of interesting problem variants not explored by our work. We list several here.

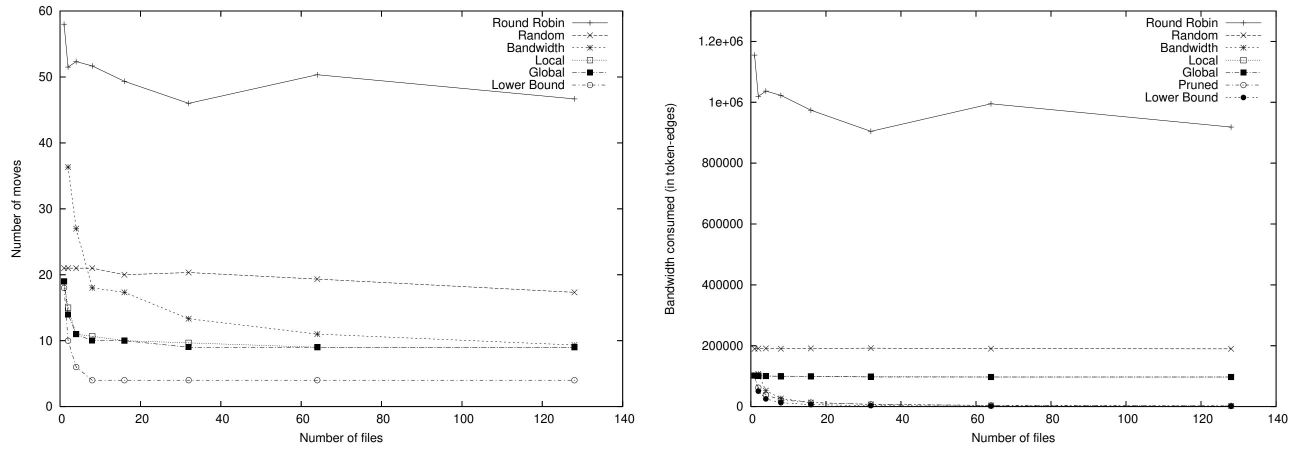
Changing network conditions. We can consider that the capacity of each arc, or even the set of arcs themselves changes between turns. By restricting the types of possible changes,



(a) Moves

(b) bandwidth

Figure 4: Moves and Bandwidth as a function of file density. Single source and file to subset of receivers on a random GT-ITM graph.



(a) Moves

(b) Bandwidth

Figure 5: Moves and Bandwidth as a function of number of files. All receivers want exactly one file subdivided from the same set of tokens. Run on a random GT-ITM graph.

this could model cross traffic, dynamic channel conditions, intermittent mobility, or even denial-of-service attacks. One interesting scenario would be to construct an on-line algorithm robust to adversarial network conditions and to compare its behavior to one with access to a network oracle that has perfect knowledge of current and future network conditions.

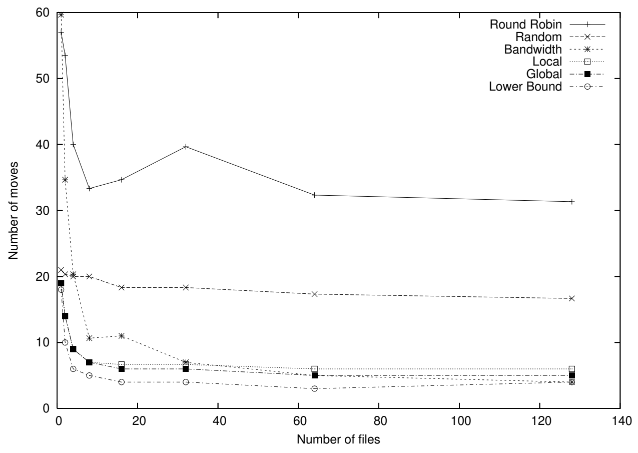
Encoding. In our problem, we consider a static set of tokens. While we admit duplication of tokens, no new types of tokens are minted within the network. In the face of lossy channels, it may be useful to introduce redundancy into the system by generating multiple sub-tokens, only a subset of which are necessary to reconstruct the original token. While such coding of the content could introduce significant additional degrees of freedom in formulating viable solutions, determining bounds may become more difficult as well.

Arrivals and departures. In any real system, participants are unlikely to join simultaneously. Instead, the set of nodes in the network will vary with time, and the content schedule should adapt to the changing distribution of token demand. This variant may be viewed as an instance of the "Changing network conditions" with capacities to and from particular nodes going from zero to non-zero and back depending on whether a node is arriving or departing.

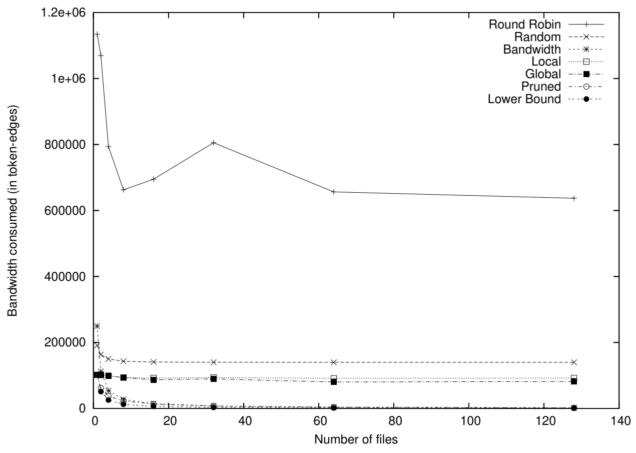
Realistic topologies. In our work, we consider only the overlay topology, and not the physical links making up our logical links. We are likely ignoring the reality that many of our logical links share the same physical link, hence their capacities are not independent. To properly model this, we need to take into account physical links and routers, which do not participate in overlay forwarding, instead simply forwarding the packets along to a specified overlay node.

Acknowledgments

This paper benefited greatly from discussions with Dejan Kostić, Kirill Levchenko, Sriram Rambahadran, and Kashi Vishwanath. We are indebted to Michelle Panik for her editorial assistance.



(a) Moves



(b) Bandwidth

Figure 6: Moves and Bandwidth as a function of number of files. All receivers want exactly one file subdivided from the same set of tokens sourced at random vertices. Run on a random GT-ITM graph.

7. REFERENCES

[1] AWERBUCH, B., AZAR, Y., AND PLOTKIN, S. Throughput competitive online routing. In *IEEE Symposium on the Foundations of Computer Science* (1993), pp. 32–40.

[2] Bittorrent. http://bitconjurer.org/BitTorrent.

[3] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., NANDI, A., ROWSTRON, A., AND SINGH, A. Splitstream: High-bandwidth Content Distribution in Cooperative Environments. In *Proceedings of the 19th ACM Symposium on Operating System Principles* (October 2003).

[4] CHERKASOVA, L., AND LEE, J. FastReplica: Efficient Large File Distribution within Content Delivery Networks. In *4th USENIX Symposium on Internet Technologies and Systems* (March 2003).

[5] CHU, Y., RAO, S. G., SESHAN, S., AND ZHANG, H. Enabling conferencing applications on the internet using an overlay multicast architecture. In *ACM SIGCOMM* (Aug. 2001).

[6] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[7] GOEL, A., HENZINGER, M. R., PLOTKIN, S., AND TARDOS, E. Scheduling data transfers in a network and the set scheduling problem. *J. Algorithms 28*, 2 (2003), 314–332.

[8] HWANG, F. K., RICHARDS, D. S., AND WINTER, P. *The Steiner Tree Problem*, vol. 53. North Holland, 1992.

[9] JANNOTTI, J., GIFFORD, D. K., JOHNSON, K. L., KAASHOEK, M. F., AND JAMES W. O'TOOLE, J. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of Operating Systems Design and Implementation (OSDI)* (October 2000).

[10] KOSTIĆ, D., BRAUD, R., KILLIAN, C., VANDEKIEFT, E., ANDERSON, J. W., SNOEREN, A. C., AND VAHDAT, A. Maintaining high bandwidth under dynamic network conditions. In *Proceedings of the USENIX Annual Technical Conference* (Anaheim, CA, Apr. 2005).

[11] KOSTIĆ, D., RODRIGUEZ, A., ALBRECHT, J., AND VAHDAT, A. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proceedings of the 19th ACM Symposium on Operating System Principles (SOSP)* (Bolton Landing, NY, Oct. 2003).

[12] PADMANABHAN, V. N., WANG, H. J., AND CHOU, P. A. Resilient Peer-to-Peer Streaming. In *Proceedings of the 11th ICNP* (Atlanta, Georgia, USA, 2003).

[13] PHILLIPS, C., STEIN, C., AND WEIN, J. Task scheduling in networks. *SIAM J. Discrete Math 10*, 4 (Nov. 1997), 573–598.

[14] ROWSTRON, A., AND DRUSCHEL, P. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Middleware'2001* (November 2001).

[15] ROWSTRON, A., KERMARREC, A.-M., CASTRO, M., AND DRUSCHEL, P. SCRIBE: The Design of a Large-scale Event Notification Infrastructure. In *Third International Workshop on Networked Group Communication* (November 2001).

[16] SHERWOOD, R., BRAUD, R., AND BHATTACHARJEE, B. Slurpie: A Cooperative Bulk Data Transfer Protocol. In *Proceedings of INFOCOM* (2004).

[17] SLEATOR, D. D., AND TARJAN, R. E. Amortized efficiency of list update and paging rules. *Communications of the ACM 29*, 2 (1985), 202–208.

[18] SNOEREN, A. C., CONLEY, K., AND GIFFORD, D. K. Mesh based content routing using XML. In *Proceedings of the 18th ACM Symposium on Operating System Principles (SOSP)* (Banff, Canada, Oct. 2001).

[19] YOUNG, A., CHEN, J., MA, Z., KRISHNAMURTHY, A., PETERSON, L., AND WANG, R. Y. Overlay mesh construction using interleaved spanning trees. In *IEEE INFOCOM* (2004).

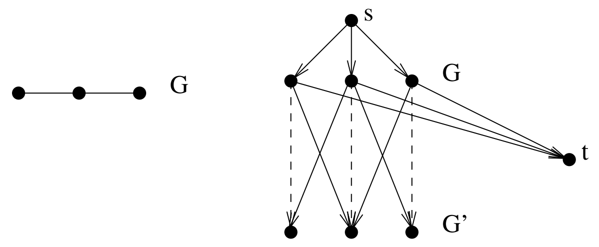


Figure 7: A graph in the Dominating Set problem and the corresponding reduction to FOCD.

APPENDIX

THEOREM 5. *The Fast Overlay Content Distribution problem is NP-hard.*

PROOF. We give a reduction from the Dominating Set problem, which is known to be NP-complete [6]. Briefly, the Dominating Set problem is to determine, given a graph $G = (V, E)$ and integer k, whether there exists a set $D \subseteq V$ of size at most k such that every vertex in $V - D$ is adjacent to some vertex in D.

Given $G = (V, E)$ and k, we construct a FOCD problem which will decide whether the graph G has a dominating set of size at most k. Let $n = |V|$. We will distribute two files, consisting of tokens $\{0\}$ and $\{1, 2, \ldots, n - k\}$. The graph for file distribution will consist of $2n + 2$ vertices; if $V = \{v_1, \ldots, v_n\}$, let the vertices in the file distribution problem be called $\{s, t, v_1, \ldots, v_n, v'_1, \ldots, v'_n\} = \{s, t\} \cup V \cup V'$.

At the start, vertex s contains copies of all tokens and no other vertex has any tokens. Vertex t wants tokens $\{1, \ldots, n-k\}$, and every other vertex v'_i wants $\{0\}$. The vertices in V act as intermediaries in distributing the tokens. There are arcs $s \rightarrow v_i$ of capacity one for each $v_i \in V$, and similarly arcs $v_i \rightarrow t$ for each v_i. Finally, there is an arc $v_i \rightarrow v'_i$ for every $v_i \in V$, and $v_i \rightarrow v'_j$ for each arc $(v_i, v_j) \in E$. This reduction is illustrated in Figure 7.

This gives a mapping reduction from the Dominating Set problem to FOCD; There is a dominating set of size k if and only if the FOCD problem can be solved in two timesteps. Note that if G has a dominating set D of size k, then the file distribution problem may be solved in two time steps by sending the vertices of the dominating set D token 0 on the first time step, and the remaining $(n - k)$ vertices in $V - D$ the tokens $\{1, \ldots, n-k\}$. On the second timestep, the tokens $\{1, \ldots, n - k\}$ may be sent to t, and each vertex of $v'_i \in V'$ may receive 0 from either v_i if $v_i \in D$, or $v_j \in D$ where $(v_i, v_j) \in E$.

Conversely, if it is possible to distribute all tokens in two steps, then $(n - k)$ vertices from V must receive tokens $\{1, \ldots, n - k\}$ in order to relay them to t, and so at most k vertices from V receive 0. Since these k vertices must send token 0 to all of V', they must correspond to a dominating set. $\square$