Swarming on optimized graphs for n-way broadcast*

Georgios Smaragdakis† gsmaragd@cs.bu.edu

Nikolaos Laoutaris‡ nlaout@eecs.harvard.edu

Pietro Michiardi§ Pietro.Michiardi@eurecom.fr

Azer Bestavros† best@cs.bu.edu

John W. Byers† byers@cs.bu.edu

Mema Roussopoulos¶ mema@ics.forth.gr

***Abstract*—In an n-way broadcast application each one of n overlay nodes wants to push its own distinct large data file to all other n-1 destinations as well as download their respective data files. BitTorrent-like swarming protocols are ideal choices for handling such massive data volume transfers. The original BitTorrent targets one-to-many broadcasts of a single file to a very large number of receivers and thus, by necessity, employs an almost random overlay topology. n-way broadcast applications on the other hand, owing to their inherent n-squared nature, are realizable only in small to medium scale networks. In this paper, we show that we can leverage this scale constraint to construct optimized overlay topologies that take into consideration the end-to-end characteristics of the network and as a consequence deliver far superior performance compared to random and myopic (local) approaches. We present the Max-Min and Max-Sum peer-selection policies used by individual nodes to select their neighbors. The first one strives to maximize the available bandwidth to the slowest destination, while the second maximizes the aggregate output rate. We design a swarming protocol suitable for n-way broadcast and operate it on top of overlay graphs formed by nodes that employ Max-Min or Max-Sum policies. Using trace-driven simulation and measurements from a PlanetLab prototype implementation, we demonstrate that the performance of swarming on top of our constructed topologies is far superior to the performance of random and myopic overlays. Moreover, we show how to modify our swarming protocol to allow it to accommodate selfish nodes.**

I. Introduction - Outline

Motivation: The BitTorrent protocol [1] has established *swarming*, *i.e.*, parallel download of a file from multiple peers with concurrent upload to other requesting peers, as one of the most efficient methods for multicasting bulk data. A fundamental characteristic of the existing BitTorrent is that the overlay graph resulting from its bootstrap and choke/unchoke algorithms is mostly ad-hoc, in the sense that it is the outgrowth of random choices of neighboring peers. This is justified given the scale of P2P file swapping networks.

P2P file swapping, is not the "be all and end all" for swarming. In this work we consider n-way broadcasting — another class of applications, in which each one of n overlay nodes must push a very large chunk of data (a distinct file) to all other $n-1$ peers, as well as pull the $n-1$ files pushed by these other peers. Once completed, this push-pull cycle may be repeated with new sets of files.

† Computer Science Dept, Boston University, Boston, MA.

‡ School of Engineering and Applied Sciences, Harvard University, Cambridge, MA.

§ Networking and Security Dept, Institut Eurecom, Sophia-Antipolis, France.

¶ Institute of Computer Science, FORTH, Crete, Greece.

* This work is supported in part by a number of NSF awards, including CNS CSR #0720604, CNS Cybertrust #0524477, CNS NeTS #0520166, CNS ITR #0205294, and EIA RI #0202067, CAREER Grant #0446522, and by the Integrated Project CASCADAS (FET Proactive Initiative, IST-2004-2.3.4 Situated and Autonomic Communications) within the 6th IST Framework Program.

Applications using n-way broadcasting would involve small/medium-sized networks, as they are inherently of n^2 nature. Examples include: distribution of large scientific data-sets in grid computing, distribution of large traffic log files for network-wide distributed intrusion/anomaly detection schemes [2], synchronization of distributed databases [3], and several other enterprise applications. Contrary to the prevailing assumption underlying the design of BitTorrent, the nodes that make up such networks are basically cooperative (at an extreme case they belong to the same administrative authority).

Even for relatively small networks, n parallel broadcasts of distinct large files can create data volumes that are impossible to handle via centralized solutions: uploading each file to a centralized server and then copying it back to all destinations in a point-to-point manner means that the same file is transmitted $O(n)$ times over the same link, *i.e.*, imposing an $O(n)$ stress on the physical links.

n-way broadcast via swarming: Swarming is clearly an attractive approach to supporting n-way broadcast applications. The obvious solution is to outsource the push-pull functionality to BitTorrent: set-up n different torrents, each one seeded by a different node.

In this paper, we question the effectiveness of BitTorrent for n-way broadcasting (which is not what it is primarily designed to support). In particular, we note that BitTorrent runs on the topologies that result from the composition of its bootstrapping and choke/unchoke algorithms. These topologies are mostly unoptimized. Indeed, the only topological optimization in BitTorrent is a local one: under the choke/unchoke algorithm, fast peers are matched up with other fast peers *from within the same randomly bootstrapped neighborhood*. By virtue of the relatively small size of neighborhoods compared to the entire network, the resulting topology is close to being random. While randomly-bootstrapped graphs may possess desirable theoretical properties (such as small diameters), they are likely to be inefficient when compared to graphs that are systematically constructed to optimize a specific application. Notice that BitTorrent's matching of fast nodes is mostly in the protocol as an efficient tool against free-riding, rather than as a conscious attempt to optimize the overall overlay topology for applications such as n-way broadcast.

Our work — Swarming over optimized overlays: For n-way broadcast applications (as well as for other potential classes of applications), the overriding goal is to optimize the efficiency of the entire overlay as opposed to creating a tit-for-tat environment to reign in selfish, free-riding behavior of individual nodes. Also, the scale of the applications we envision makes it possible/practical to optimize the construction of the overlay, especially if distributed optimization is used.

Armed with this realization, our goal will be to construct highly efficient topologies to be used by swarming protocols for n-way broadcast. Specifically, we construct an optimized, common overlay network, upon which swarming is used. In order to control the stress of the physical links supporting the overlay, we impose an upper bound on the degree of the nodes in the constructed overlay network.

Next we present justification for several of the salient features of our solution – features that will be developed and presented fully later in the paper.

Why swarming on top of an overlay? Because hop-by-hop relay of the entire file over a shortest-path tree embedded on the overlay topology and rooted at the seed node would take too long. We want to harness the power of parallel downloads as exemplified in BitTorrent.

Why use a common overlay? Because a topological optimization requires monitoring the performance of overlay links, and we want to amortize the cost of such monitoring — pay it only once per link and reuse the result for the benefit of all n transmissions (and avoid monitoring the same link up to n times as can happen if one builds n independent overlays).

How could swarming benefit from an end-to-end optimized overlay? Our overlays are optimized for end-to-end performance over multi-hop paths, *e.g.*, by maximizing the minimum available bandwidth to any destination over multiple paths, or by maximizing the total available bandwidth to all destinations over all available paths. From a single node's perspective, swarming involves point-to-point transfers within the neighborhood of that node. Each node, however, has in its neighborhood nodes that also belong to other "adjacent" neighborhoods. Noting this, one can see that, through swarming, data chunks eventually reach their destinations through multi-hop paths formed through single hop transfers between neighborhoods. If these multi-hop paths are end-to-end optimized, then swarming will be more effective in operating upon them as compared to upon unoptimized paths.

Why optimize the overlay based solely on network characteristics, without consideration of data availability? Arguably, one could conceive of more general overlay constructions in which neighbors are selected based on criteria involving both the network characteristics and the availability of chunks at each candidate connection point. In our work, we adopt a bandwidth-centric/data-agnostic approach to the construction of the overlay for two main reasons: (1) for large objects it is high bandwidth that leads to small delivery completion times and high object throughput; (2) the global state in terms of available chunks per node changes too frequently (with each successful chunk exchange between two nodes), resulting in an optimized topology that changes too frequently to be of practical use. The fact that we do not consider data availability in the construction of the overlay does not mean that data availability does not play a role in our approach: it does, but not at the overlay construction time-scale. Specifically, *we advocate a "two-pronged approach" operating at two distinct time scales*: at a coarse time scale, we address issues related to network characteristics through the construction of a dynamic, distributively optimized overlay, and at a finer time scale, we address issues related to data availability through the upload/download scheduling algorithms employed in the swarming protocol that runs on top of the overlay.

II. Related Work

This work is the fusion of two very recent thrusts in networking research: *network creation games* and *swarming protocols*. Network creation games appeared in computer science with the work of Fabrikant et al. [4] in which a set of nodes forms a network in a distributed manner driven by self-interest — each node pays for the creation of a number of links to other "neighbors" so as to minimize a hybrid cost that captures the purchase cost of these links and the delay for routing packets to all other destinations using own and remote links. The model targeted the creation of physical telecommunication networks through peering agreements between ISPs (hence the explicit modeling of the cost of buying a link). Laoutaris et al. [5], studied the "capacitated" version of the above problem, targeting the construction of overlay routing networks — each node is given a bound on the number of immediate peering relationships that it can establish (defined by the protocol that implements such an overlay network) and selects the best neighbors so as to minimize its sum of distances to all destinations through shortest-path routes over the resulting overlay topology. These works differ fundamentally from ours in that they target *routing*, *i.e.*, they assume that a packet from v to u is of interest only to u. Intermediate nodes w that lay on the overlay path from v to u are there just to assist in the routing of the packet. In the current work, each node is *broadcasting* a file to all destinations and thus intermediate nodes are also receivers in addition to being relay points. More fundamentally, in our case the delivery of information from v to u occurs not through a single path but (potentially) through all the available connected paths between the two end-points (because the file is cut into chunks which travel in parallel along different paths on the overlay). For this reason we employ max-flows as building blocks for designing the overlay (as opposed to shortest-paths which are used in point-to-point routing [4], [5]). Max-flows reflect better the nature of our application (broadcasting) as well as the nature of the employed technique for implementing it (swarming).

The BitTorrent protocol [1] has established swarming as one of the most fresh and promising ideas in contemporary networking research and thus has kicked-started a (tidal) wave of research articles in the area. Our fundamental difference from this body of work, whether analytic, e.g., Qiu and Srikant [6], Massoulie and Vojnovic [7], Kumar and Ross [8], experimental, e.g., Bharambe [9], or measurement based, e.g., Izal et al. [10], Legout et al. [11], is that we have substituted the (close to) random graph resulting from BitTorrent's bootstrap and choke/unchoke algorithms with a highly efficient distributively optimized graph. As we show later on, such a switch boosts the performance of a swarming protocol running on top of it. We are able to obtain such highly efficient graphs because our interest is on smaller networks. We show that at such scales one can do much better than close to random.

Some other relevant works are the following ones. Massoulie et al. [12] recently showed that a simple distributed randomized algorithm can achieve the theoretical optimal broadcast rate given by Edmond's theorem [13] for a source node in a flow network. Compared to this work, we let each node select its neighbors and thus participate in the construction of the flow network, as opposed to taking it for granted. Gkantsidis and Rodriguez [14] have proposed the use of network coding as an alternative to BitTorrent's chunk scheduling algorithm. The performance benefit/added complexity ratio of employing network coding is not yet generally agreed upon [11]. Although we focus on BitTorrent-like swarming here, our optimized topologies should also benefit network-coding based swarming because they are oblivious to whether network coding is used or not.

Guo et al. [15] and Tien et al. [16] look at the design of multi-torrent systems. Their contribution is mostly on the measurement and the design of inter-peer incentive mechanisms for peers that participate in multiple torrents concurrently. They do not look at overlay construction issues. Interestingly, Tien et al. [16] provide justification for one of our design choices, which is to enforce that at any time there should be only one active torrent between any two nodes (more in Sect. IV). They show that deviating from this choice and allowing transferring between two nodes multiple chunks in parallel (one for each torrent), slows down the system by over-partitioning the upload bandwidth of nodes.

Other end-system multicast systems such as SplitStream from Castro et al. [17] and Bullet from Costic et al. [18] could be used to support n-way broadcasting by creating a separate overlay for each source. The problem with this approach is that there is no coordination across different overlays and thus there can be performance inefficiencies as well as significant overheads due to the redundant monitoring of the same physical paths multiple times from different overlays. Our approach is to construct one overlay for all sources and thus jointly optimize as well as share the monitoring cost.

The only work we are aware of on the intersection of overlay creation and BitTorrent is a very recent one from Zhang et al. [19]. It looks at the formation of Nash equilibria topologies in view of download-selfish peers that participate in a single torrent. Our overlay formation, although distributed and based on local utility functions is: (1) primarily targeting the optimization of the social utility of the network, meaning that all nodes are assumed to be under common control, and (2) considering both upload and download performance for multiple torrents, one at each node. We examine selfishness issues and how these could be addressed towards the end of our article, but this is just a supplement of our main contribution.

III. PEER-SET SELECTION

Let $V = \{v_1, v_2, \ldots, v_n\}$ denote a set of nodes. Node v_i selects k other nodes to be in its *peer-set* $s_i = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ and establishes bidirectional links to them. Let $S = \{s_1, s_2, \ldots, s_n\}$ denote the edge set of the *overlay graph* $G = (V, S)$ resulting from the superposition of the individual peer-sets. Each link of G is annotated with a capacity c_{ij} which captures the available bandwidth [20] (availbw) on the the underlying IP layer path that goes from v_i to v_j. Capacities can be asymmetric, meaning that $c_{ij} \neq c_{ji}$ in the general case. Let $MF(v_i, v_j, S)$ denote the resulting max-flow from v_i to v_j under S. Let also $\Phi(v_i, S)$ and $\Psi(v_i, S)$ denote the minimum max-flow from v_i to any other node under S, and the sum of max-flows from v_i to all other nodes under S, respectively, *i.e.*:

$$\Phi(v_i, S) = \min_{v_j \in V_{-i}} MF(v_i, v_j, S), \Psi(v_i, S) = \sum_{v_j \in V_{-i}} MF(v_i, v_j, S)$$

In the above definitions, each max-flow from v_i to an individual destination is computed independently of other max-flows from the same node to different destinations (*i.e.*, each one is computed on an empty flow network G). These definitions should not be confused with multi-commodity flow problems in which multiple distinct flows co-exist.

Definition 1: (Max-Min and Max-Sum peer-sets) A peer-set s_i is called Max-Min if it maximizes the minimum max-flow of node v_i, *i.e.*, $\Phi(v_i, \{s_i\} + S_{-i}) \geq \Phi(v_i, \{s_{i'}\} + S_{-i})$, $\forall s_{i'} \neq s_i$, where S_{-i} denotes the superposition of the peer-sets of all nodes but v_i. Similarly, a peer-set is called Max-Sum if $\Psi(v_i, \{s_i\} + S_{-i}) \geq \Psi(v_i, \{s_{i'}\} + S_{-i})$, $\forall s_{i'} \neq s_i$.

Lemma 1: Finding a Max-Min or Max-Sum peer-set for v_i given S_{-i} is an NP-hard problem.

Proof: See Appendices A and B. ∎

These peer-set selection policies optimize the connectivity of a given node to the remaining network. One could say that this constitutes selfish behavior. This is indeed the case if the nodes use this connectivity to *only* disseminate their own file. However, when they also indiscriminately relay the files of others, which is the assumption for the applications we consider, then optimizing one's connectivity boosts the aggregate social performance of the network. Later on, in Sect. VI we discuss what happens when the swarming protocol (running above the overlay) ceases to be indiscriminate with respect to the upload quality it gives to local and remote files.

Why Max-Min and Max-Sum? Given a flow network G, the *broadcast problem* asks what is the maximum (broadcast) rate at which a source v_i can deliver its stream concurrently to all other nodes. Edmonds showed in [13] that the broadcast rate is equal to $\min_{v_j \in V_{-i}} \mathrm{mincut}(v_i, v_j)$, which in view of the max-flow/min-cut theorem is equal to $\min_{v_j \in V_{-i}} MF(v_i, v_j)$. Therefore, the Max-Min peer-set is the peer-set that maximizes the broadcast rate of a node, or conversely the delivery rate to the slowest receiving peers. It does so by placing the links so as to boost the max-flow to these slowest peers. Of course for this to be possible there must be available bandwidth to be utilized at the IP level (this is reflected on the c_{ij}'s which steers the peer-set selection, and which are obtained through measurements as explained in Sect IV). Edmonds gave an exponential time centralized algorithm for achieving the broadcast rate, which was later improved to a small polynomial time by Lovasz, Gabow and others [21]. Recently, Massoulie et al. [12] showed that a simple randomized decentralized algorithm can achieve a delivery rate that is arbitrarily close to the broadcast rate.

A Max-Sum peer-set on the other hand is a peer-set that maximizes the *theoretical maximum aggregate transmission rate* from a node. Contrary to the Max-Min peer-set that maximizes a provably attainable broadcasting rate, the Max-Sum maximizes only an upper bound on the aggregate rate which, in the general case, is not attainable due to contention for link bandwidth when max-flows from the same source to different destinations share common overlay links.[1] We elaborate with an example.

Consider the flow network of Fig. 1 (top-left) in which all links have unit capacity and node 1 is the source. Computing each max-flow on an empty network we get that the max-flow from the source to nodes 2, 3, and 4 is equal to 1 whereas that to nodes 5 and 6 is equal to 2, thereby $\Psi(1) = 7$. Consider now the maximum real flows that can exist concurrently from the source to nodes 5 and 6 (top-center). Breaking the file into two equal parts A and B the source can transmit A at full rate over the dotted paths ($1 \rightarrow 2 \rightarrow 5$ and $1 \rightarrow 4 \rightarrow 6$) and B at full rate over the dashed path (only once over link (1,3)) and achieve concurrent real flows that match the capacity of corresponding max-flows on an empty graph, *i.e.*, RF(1,5)=MF(1,5)=2 and RF(1,6)=MF(1,6)=2. This is possible because a single transmission of B on the edge (1,3) suffices for contributing to both RF(1,5) and RF(1,6). Thus the two flows don't compete for bandwidth on the shared link and can achieve the same capacity as the corresponding max-flows on empty networks. This is not, however, generally possible. On the top-right part of the figure we depict the situation when sending from the source to all destinations (nodes 2-6) concurrently. In this case the entire file (both A and B) has to go over links (1,2), (1,3), and (1,4) and thus RF(1,5)=RF(1,6)=1<MF(1,5)=MF(1,6)=2 leading to a real aggregate rate $\tilde{\Psi}(1) = 5$ smaller than the bound $\Psi(1) = 7$.

Generally, the bound becomes less tight with increasing link density k/n. On the bottom-left part of Fig. 1 we add to the previous network two new links: (3,2) and (3,4). It is easy to verify that the max-flow from the source to nodes 2, 4, 5, and 6 is now 2 and to node 3 is 1, leading to $\Phi(1) = 9$. As before, if we consider only the flows to 5 and 6, it is easy to see that their max-flow values can co-exist. Considering, however, the flows to all destinations, we see that any partition of the file into parts will inevitably lead again to all real flows being 1, whereas the corresponding max-flows with the exception of MF(1,3) are now 2.[2] In other words, although the new links increased both MF(1,2) and MF(1,4) by 1 compared to the previous network, they cannot increase any of the real flows and thus widen the gap between the bound ($\Phi(1) = 9$) and the maximum attainable aggregate rate ($\tilde{\Phi}(1) = 5$).

To sum up, we propose and study these peer selection policies for the following reasons: (1) Max-flows are used to capture the fact that in a swarming protocol the chunks of a source node v_i travel towards a sink node v_j over (potentially) all the available paths of the overlay graph of point-to-point peer relationships. (2) The gap between the bound on the aggregate rate $\Psi(v_i, S)$ given by a Max-Sum peer-set and the actual maximum attainable aggregate rate $\tilde{\Psi}(v_i, S)$ which factors in the sharing of overlay links from multiple max-flows

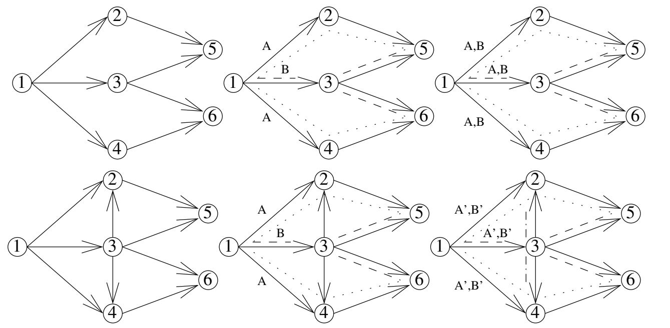

Fig. 1. Mixing max-flows. Left: empty network. Middle: RF(1,5) and RF(1,6) co-existing. Right: RF(1,2), RF(1,3), RF(1,4), RF(1,5), RF(1,6) co-existing. Top: initial network. Bottom: Initial augmented with edges, (3,2) and (3,4).

to different destinations, is reduced by the fact that swarming protocols guarantee that any chunk is transmitted at most once between any two peers; therefore, $\tilde{\Psi}(v_i, S)$ can use an overlay link multiple times (for different max-flows) but would seize bandwidth only once, thereby reducing its gap from the bound $\Psi(v_i, S)$ that assumes that the entire flow network is available to each individual max-flow from v_i. (3) The overlay network has to be rather sparse (small k) so as to limit the stress on the physical links. Thus the bound Max-Sum won't be very much off from the actual achievable aggregate rate and it makes sense optimizing the peer-set based on it. Regarding Max-Min, this is provably attainable, and optimal for broadcast rate as discussed earlier.

Since a node cares to both upload its local file to all other nodes as well as download from them all remote files, we combine the previous definitions in the following objective functions:

$$\dot{\Phi}(v_i, s_i) = \alpha\Phi(v_i, \{s_i\} + S_{-i}) + (1-\alpha) \min_{v_j \in V_{-i}} MF(v_j, v_i, \{s_i\} + S_{-i}),$$
$$\dot{\Psi}(v_i, s_i) = \alpha\Psi(v_i, \{s_i\} + S_{-i}) + (1-\alpha) \sum_{v_j \in V_{-i}} MF(v_j, v_i, \{s_i\} + S_{-i})$$

In the above functions, the parameter α regulates the relative importance between upload and download quality in selecting a peer-set. If the link capacities are symmetric, then optimizing $\dot{\Phi}$ or $\dot{\Psi}$ reduces to optimizing Φ or Ψ, independently of α.

IV. Node Architecture

Nodes consist of the following components: a *peer selection module* implementing the peer-set selection algorithms described in Sect. III; a *downloader module*, responsible for issuing requests to neighboring nodes and downloading missing chunks; and an *uploader module*, responsible for sending back local and in-transit chunks (an in-transit chunk is a chunk that does not belong to the local source file). In this section we describe these three modules under the assumption that nodes are cooperative (therefore we don't need mechanisms like choke/unchoke). Later on, in Sect. VI we discuss the necessary changes for dealing with selfishly behaving nodes.

A. Peer Selection Module

Every time period T, a node: (1) measures its available bandwidth to all other nodes using `pathChirp` [22], (2) executes a peer-set selection algorithm from Sect. III and connects to the corresponding nodes (incoming links are left untouched). Since both Max-Min and Max-Sum are NP-hard, we use fast local-search heuristics to compute approximately

[1] The contention between max-flows "from" different sources does not come explicitly in these objective functions. It is captured in our framework through the measured availbw c_{ij}: the availbw on a direct overlay link from v_i to v_j depends on the capacity of the underlying physical path *and the amount of this capacity already captured by the competing max-flows from other sources. At this level the problem is indeed a multi-commodity flow.*

[2] The fact that the entire file has to go over the edge (1,3), eliminates any chance for increasing the real flows to nodes 2, 4, 5, and 6 beyond 1.

optimal peer-sets (which we verified to be always within 1% of the exact optimal for all problem sizes on which we were able to use integer linear programming to compute the latter). Once links are established, the node keeps monitoring them (including the incoming ones) and relays their capacity to all other nodes through an overlay link-state announcement protocol. Remote nodes need this information to compute their own peer-sets. Although each node measures $O(n)$ overlay links every re-wiring epoch T, the monitoring and announcement overhead is only $O(kn)$ and not $O(n^2)$ since only the $O(k)$ established links are monitored and announced in between the (infrequent) rewiring epochs, where $k \ll n$.

B. The Downloader Module

The downloader module monitors the available chunks on the peer-set and issues requests for downloading missing ones. The selection is based on the well established *Local Rarest First* (LRF) heuristic [11] that looks at the peer-set and issues a request for any missing chunk that is among the least replicated ones in the peer-set. New requests are triggered either upon the completion of a download, or if an overlay link is inactive, upon the detection on the other side of the link of a missing chunk.

C. The Uploader Module

The uploader receives requests and sends back chunks. Our baseline uploader allows for *up to 1 active upload (chunk) per overlay link (neighbor)*. It implements this by maintaining a FIFO queue for each overlay connection. This choice bounds the number of concurrent uploads by the number of neighbors thereby avoiding excessive fragmentation (over partitioning) of the upload bandwidth of the local (physical) access link of a node (this choice is backed-up by results appearing in [16]). We also experimented with an uploader that allows up to 1 active chunk per source file per connection, but this can lead to up $n-1$ parallel uploads per overlay link, which becomes problematic as n increases. Indeed, over-partitioning the upload bandwidth defeats the entire concept of swarming: it takes too much time to upload an entire chunk, and during this time the downloading node is under utilizing its upload bandwidth as it cannot relay the chunk before it completes the reception. We want to note, however, that our baseline design is by no means claimed to be optimal. For an example consider a node that can upload to its first $k-1$ neighbors with rate x and to the last one with rate larger than $k \cdot x$. Then as long as this last neighbor can always find k missing chunks from our node, and can also itself disseminate them further down in the network faster than the $k-1$ slow neighbors, then the system would be better off allowing up to k parallel uploads to the fast one at the expense of the slow ones. Such situations though are rather peculiar and even if they arise, it is difficult to check the necessary conditions for taking advantage of them, so we leave their investigation to future work and stick to the simple one-chunk-per-connection policy.

V. PERFORMANCE EVALUATION

In this section we compare the performance of Max-Min and Max-Sum peer selection policies against three reference selection policies: Random (node v_i selects k peers at random from the set of all nodes in V_{-i}); k-Widest (node v_i selects node v_j if c_{ij} is among the k largest ones across all nodes in V_{-i}); Rand k-Widest (v_i performs k-Widest on a random subset of V_{-i} of size $\beta \cdot k$). Rand k-Widest is included in the evaluation to mimic the effect of combining random bootstrapping with choke/unchoke in BitTorrent.

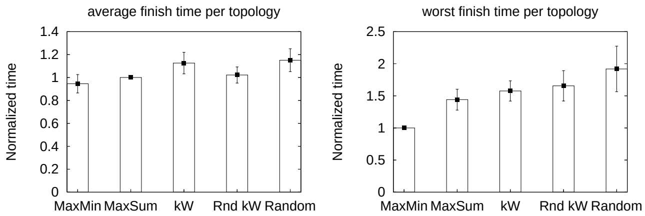

Fig. 2. PlanetLab experiment, mid June 2007.

We compare these policies in terms of (node,remote file) *finish times*. We denote $f(j,i)$ the time that the sink v_j completes downloading the file of source v_i, assuming that all exchanges start at time 0. In all experiments we assume that nodes are fully cooperative (they belong to the same authority) and thus follow exactly and truthfully the peer-selection policies of Sect. III and the swarming protocol of Sect. IV (*i.e.*, no choke/unchoke mechanism is employed). We discuss the impact of selfishly behaving nodes in Sect. VI.

Our performance evaluation is done in two settings. In the first, we assume that the n-way broadcast is to be carried over the Internet. We do so by evaluating the performance of a prototype implementation of our architecture on PlanetLab. In the second, we assume that the n-way broadcast is to be carried on a closed (controlled/isolated) network. We do so by evaluating the performance of a prototype implementation of our architecture on a discrete event simulator of the closed network. [23] contains additional results and discussion that do not fit here.

A. Case Study 1: A PlanetLab Prototype

In this setting, we compare the performance of different overlay topologies when the underlying physical network is the Internet and the overlay nodes are single-homed, *i.e.*, all overlay links of a node go over the same physical access link. For this purpose we selected $n = 15$ PlanetLab nodes (10 in North America, 1 in South America, 3 in Europe, 1 in Asia) and let each one of them disseminate a different 100MBytes file and allow it to connect to $k = 2$ neighbors (and accept additional incoming links). Notice that we limited our experiment to only 15 nodes and only 100MBytes per node so as to keep the amount of exchanged traffic on PlanetLab at reasonable levels, while also allowing us to monitor the network throughout the experiment. Notice that if data were to be transfered in a point-to-point manner, then it would amount to over a Terabyte for each execution of the entire experiment: 5 different peer-set selection policies, each one generating 15·14·100MBytes of data at each run, and repeated 10 times to get confidence intervals. We let the re-wiring epoch be $T = 10$ minutes and the measurement/announcement epoch for existing links be 2 minutes. Also we set $\alpha = 0.5$ to indicate that nodes care equally for download and upload quality.

For a node v_j, we compute its maximum finish time $\max(j) = \max_{i \neq j} f(j,i)$, *i.e.*, the time at which it has completed downloading *all* $n-1$ remote files, as well as its average finish time $\text{avg}(j) = 1/(n-1)\sum_{i \neq j} f(j,i)$. For peer-set selection policy $\mathcal{X}$, we let $\max(\mathcal{X}) = \max_j \max(j)$ denote its *maximum finish time* across all nodes, and $\text{avg}(\mathcal{X}) = 1/n \sum_j \text{avg}(j)$ denote its *average finish time* across all nodes.

On the left-hand-side of Fig. 2 we present the normalized average finish time of each policy with respect to the average finish time of the Max-Sum policy. On the right-hand-side, we present the normalized maximum finish time of each policy with respect to the maximum finish time of the Max-Min policy. These results show that the various policies perform quite similarly with respect to average finish time. When looking at maximum finish times though, the picture is completely different. Max-Min manages to complete all downloads anywhere between 40% and 120% faster than the heuristics and almost 30% faster than Max-Sum. This can be very significant for Bulk Synchronous Parallel (BSP) applications [24], in which the global progress depends on the finish time of the slowest node. It is worth noting that optimizing the worst case finish time is much more difficult than optimizing the average, and thus it should come as no surprise that the heuristics perform well on average but fail to improve the worst case.

B. Case Study 2: A Dedicated Network Prototype

In this setting, we examine overlay networks whose links are dedicated, meaning that they do not compete for bandwidth on the underlying physical network. This model is plausible for (multi-homed) networks set-up in support of an enterprise through the acquisition of dedicated links to connect its various locations. Such link acquisitions could be done through SLA contracts with ISPs, or through virtualization technologies such as those envisioned for GENI. In either cases, a dedicated link could be set up between two enterprise nodes i and j for a given price. Any such dedicated link will have a nominal capacity c_{ij}, which may depend on any number of factors (*e.g.*, physical constraints of the underlying technology, the demand at the ISP for carrying traffic between these two locations, or the price paid for various links. Since setting up a complete network to connect all n nodes directly to each other may not be feasible (especially for systems of moderate sizes), designers of such enterprise networks are likely to construct the network so as to maximize its utility with respect to some objective function. Independent of which process/strategy is used to construct the optimized overlay, the resulting network would allow all enterprise nodes to communicate either directly or through overlay paths.

The construction we propose for optimizing the overlay for n-way broadcast proceeds as follows. First, we order the nodes according to their ids. Next, we proceed in rounds in which nodes take turns in selecting their peer-sets (as discussed in Sect. III). This process is repeated until we converge by reaching a round that does not introduce changes in the constructed topology.[3]

[3] It is worth noting that the convergence of the above procedure relates to a question regarding the existence of pure Nash equilibria, and their reachability through local improvement paths, in a strategic game with Max-Min or Max-Sum as its payoff function. Although interesting from a theoretical standpoint, the question is not directly relevant here as we have assumed that nodes forward indiscriminately local and in-transit chunks. In all our experiments we got fast convergence but could also stop prematurely after a maximum number of iterations so as to deal with inexistence, unreachability, or slow convergence to stable topologies.

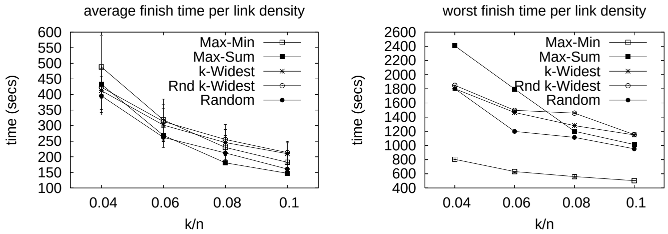

Fig. 3. Simulation of a closed network based on Sprint's topology.

Towards our goal of evaluating the impact of various peer selection policies on the performance of n-way broadcast in this setting, we developed a discrete-event simulator that is able to run over dedicated overlay networks. We constructed the dedicated overlay (enterprise) network using the procedure described above, using the publicly available trace of Sprint's physical topology taken from Rocketfuel [25]. In particular, we assumed that the dedicated capacity that could be acquired from the ISP (Sprint) would reflect an "equal-share" partitioning, which we approximated as follows. We counted the number of shortest-paths (for all physical node pairs) that go over a physical link and set the available bandwidth of that link to be its real capacity divided by this number.[4] Then, for an overlay link (i,j) we set $c_{i,j}$ to be equal to the available bandwidth of the tightest physical link on the induced shortest-path over the physical topology. This produces the amount of available bandwidth that the ISP can guarantee for the new application if it admits it into its network and treats it equally with pre-existing ones.

One advantage of simulations (compared to PlanetLab prototyping) is that it allows us to consider a bigger network. In particular, in the experiments that follow, we study overlays of size $n = 50$ nodes, which are randomly selected from the physical Sprint network — each node holding a 500Mbytes file. In Fig. 3 we compare the average and maximum finish times of different policies for different link densities k/n. Compared to the previous results from PlanetLab, we observe a qualitatively similar behavior. The gap, however, between Max-Min and the rest in terms of maximum finish time widens substantially: Max-Min is able to finish 2-3 times faster in this setting, even for relatively large k/n ($\sim$10%). The reason is that Max-Min has more real bandwidth to work with in this case: When it places a link (i,j), the capacity (both upload and download) of the two end-points increases by the capacity of the newly-added dedicated overlay link, whereas in PlanetLab the physical bandwidth is fixed, so when Max-Min places an overlay link it can only benefit by whatever unused bandwidth exists on the underlying physical network.

VI. Dealing with Selfish Behavior

Up to now we have assumed that nodes are fully cooperative, which is a realistic assumption for the applications enumerated in the introduction. In this section we will try to explore ways to accommodate applications that involve selfish nodes. We will focus on the following definition of selfishness:

[4] The idea is that each pair of physical nodes represents a different application that is assigned an equal share of the physical capacity of all links on which it competes with other applications.

Definition 2: (Upload-selfishness) An *upload-selfish* node is a node that wants to use as much of its upload capacity as possible for forwarding its local chunks and avoid "wasting" it in relaying the in-transit chunks that it holds.

A. A Brief Taxonomy of Deterrence Mechanisms

The amount of extra benefit for an upload-selfish node (and potential harm to others) depends on the mechanisms that the network deploys for discouraging such behavior. We examine the following cases.

Case 0 (neutral): Here the network stays neutral and does not deploy any deterrence mechanism. In such a setting, the upload-selfish node could simply upload its own chunks and ignore all other requests. The harm to cooperative nodes can easily be quantified for this case, so we don't discuss it further; it will be proportional to the number of upload-selfish nodes, and cooperative nodes will be slowed down and at an extreme case will be unable to receive some files (*e.g.*, when all their neighbors are upload-selfish, which is similar to the case of an eclipse attack [26]).

Case 1 (oblivious retribution): A network can employ several retribution mechanisms to punish a node that fails to deliver a chunk after a request. The choke/unchoke [1] mechanism of BitTorrent, or modified versions based on bit-level tit-for-tat [9], [15] are two established existing proposals. Contrary to the original BitTorrent, such mechanisms are marginally useful here because they are oblivious to whether a node uploads local or in-transit chunks. An upload-selfish node will appear to be contributing by the mere fact that it is certainly uploading its own chunks. Thus oblivious strategies fail to punish nodes that "free-ride" by not uploading in-transit chunks.

Case 2 (non-oblivious retribution): Now, let's assume that there exists a non-oblivious retribution mechanism that punishes a node that fails to service requests[5] for in-transit chunks that it holds. What can a selfish node do against such mechanism? The simplest strategy is to hide (by not announcing) the availability of in-transit chunks it holds, and thus get rid of the burden of having to service requests for these chunks. This can be addressed with a simple *two-hop announcement strategy* in which a node that uploads to another node announces on its behalf the availability of the chunk (using HAVE messages [1]) to downloaders belonging to the peer-set of the receiving node. This requires obtaining upon bootstrap (and re-wiring) second hop neighbors. Assuming that the retribution is severe enough, the upload-selfish node will have to honor all requests. Despite that, the upload-selfish node still has some room to game the system by changing the uploader and the downloader as follows.

– The upload-selfish node can substitute each FIFO queue at its uploader with a *selfish FIFO* (S-FIFO) that gives priority (preemptive or non preemptive) to requests for local chunks.

– The upload-selfish node can switch from Least Replicated First to Most Replicated First (MRF) downloads. Highly replicated chunks receive fewer requests and thus reduce the "waste" of upload bandwidth for sending in-transit chunks, is smaller (most nodes already have these chunks, and any requests for these chunks will be divided over many peers).

[5] We do not want to punish nodes that don't have enough in-transit content for whatever reason (slow local link or peer-set) but would relay if they had, so we only punish when a request exists and is not honored.

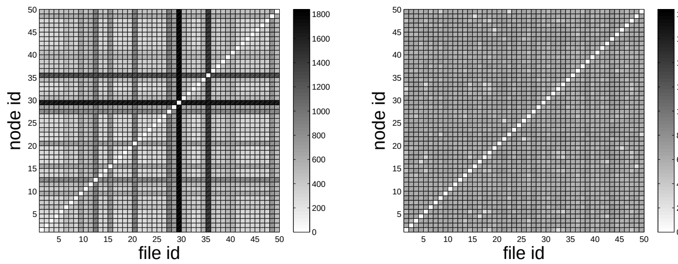

Fig. 4. Maximum finish time for all nodes and all files under Random with cooperative nodes and Max-Min with upload-selfish slowest node ($\frac{k}{n} = 0.04$).

Since it is difficult to detect such deviations from the protocol, we instead quantify their impact.

B. Quantifying the Impact of S-FIFO/MRF

We quantify the advantage for a single upload-selfish node by looking at the ratio between the time it takes to upload its file to all other nodes when it is selfish and when it is cooperative, granted that all other nodes are cooperative. We examined this ratio for different overlays built on the Sprint trace and for different choices with respect to the choice of selfish node. We consider three cases, where the selfish node is : (1) the *slowest* node, *i.e.*, the one whose adjacent links have the minimum aggregate upload capacity; (2) the *fastest* node; or (3) a *typical* node (median upload capacity).

On the Max-Min overlay the selfish node reduced its maximum upload finish time by 30% when it was the slowest one [23]. When it was a typical (or the fastest one), then it got almost no benefit, since in these cases the bottleneck is at the downloading nodes (so a local selfishness behavior cannot help). In all other overlays, the selfish node got almost no benefit, even when it was the slowest node. Unlike the Max-Min, the other overlays are not optimized for the slowest node, so even if this bottleneck node tries to selfishly upload its file, it cannot really benefit because it has very limited bandwidth.

From the above, it is clear that there exist cases in which upload-selfishness pays substantially. Granted that upload-selfishness is hard to detect, we also look at its impact on the cooperative nodes. We consider again a single selfish node (one can easily extrapolate for multiple selfish nodes). The impact depends on the considered metric and on the identity of the selfish node. If we care about the worst-case download time of cooperative nodes and let the selfish node be the slowest node, then counter-intuitively, the impact on the cooperative nodes is positive. This is simply because by being selfish, the slowest node helps all other nodes improve their (bottleneck) downloads from it. To get a feeling of this we show a scatter-plot on the left of Fig. 4 with the download time for each pair (node,remote file) when the topology is random and all the nodes are cooperative. The solid black line that stands out corresponds to the slowest node (node 29), whose file is the last one to be downloaded by all others. To contrast this, we plot on the right the corresponding times when the topology is Max-Min and the slowest node is upload-selfish. As it can be seen, the combination of Max-Min topology and upload-selfish scheduling on the slowest node does a pretty good job at smoothing out the differences in maximum finish times. If, on the other hand, the selfish node is a typical node, or the fastest node, then its effect on the download quality of others is rather marginal. First, its own file is not a bottleneck. Second, the relay of in-transit chunks is largely carried by the other $n-1$ nodes. Third, S-FIFO and MRF impact primarily first-hop neighbors and have small impact on nodes further away.

Overall, upload-selfishness, unlike its name suggest, is not necessarily bad. A socially inclining global scheduling policy, for example, would certainly make slow nodes upload only their own chunks so as to reduce the severity of the bottlenecks that they cause. More generally, for social optimality, one should split the upload bandwidth of a node between local and in-transit chunks according to the relative speed of the node. Nodes who are fast should contribute heavily in relaying in-transit chunks. Nodes who are slow, should focus only on uploading their own chunks so as to avoid becoming bottleneck points. Stated differently, *a single uploading policy across all nodes cannot be socially optimal*. We postpone the investigation of node-dependent upload scheduling for future work (see Sect. VI of [27] for a similar discussion based on our previous work on selfish caching).

C. Download-Selfishness and Other

It is tempting to ask whether a notion of *download-selfishness* would make sense. Our answer leans towards the negative. First, there is no contention between local and in-transit chunks in the incoming direction towards a node — only in-transit chunks flow there. Second, as long as the downloader keeps all its overlay connections busy by immediately identifying and requesting missing chunks, its download-finish time will be the same, so it gets no foreseeable benefit by deviating from LRF. Finally, trying to manipulate the system by advertising false c_{ij}'s for the established links can be disclosed by having nodes periodically "audit" others by measuring some remote c_{ij}'s and comparing with the advertised values on the link-state protocol. Such methods are quite elaborate and fall outside the scope of the current work.

VII. Conclusions and Future Work

In this article we showed that swarming protocols for bulk data transfers perform much better when operating over optimized overlay topologies that take into consideration the end-to-end performance characteristics of the underlying network. Such topologies improve the aggregate transmission capacity of nodes, but where they make a huge difference compared to existing heuristic approaches, is on relieving bottleneck points. Random and myopic heuristics used in practice lack the required sophistication for overcoming such bottlenecks.

Our optimized topologies are oblivious to the details of the swarming protocol that runs on top. They leverage the available bandwidth of the underlying network and abstract the swarming protocol by viewing it as a series of max-flows. Thus they can benefit a variety of swarming protocols with different upload/download scheduling characteristics. Since our topologies are data-blind, it is the job of the swarming protocol to make the best use of the end-to-end bandwidth that they offer. To that end, we have shown that a commonly parameterized swarming protocol is far from being optimal. Designing swarming protocols tailored to the characteristics of individual nodes is on our future research agenda.

References

[1] B. Cohen, "Incentives build robustness in BitTorrent," in *Proc. of First Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, 2003.

[2] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, and G. Iannaccone, "Detection and identification of network anomalies," in *Proc. of IMC '06*, Rio de Janeriro, Brazil, 2006.

[3] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Comput. Surv.*, vol. 13, no. 2, 1981.

[4] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker, "On a network creation game," in *Proc. of ACM PODC '03*, Boston, MA, 2003.

[5] N. Laoutaris, G. Smaragdakis, A. Bestavros, and J. Byers, "Implications of selfish neighbor selection in overlay networks," in *Proc. of IEEE INFOCOM '07*, Anchorage, AK, 2007.

[6] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks," in *Proc. of ACM SIGCOMM '04*, Portland, Oregon, USA, 2004.

[7] L. Massoulie and M. Vojnovic, "Coupon replication systems," in *Proc. of ACM SIGMETRICS '05*, Banff, Alberta, Canada, 2005, pp. 2–13.

[8] R. Kumar and K. W. Ross, "Optimal peer-assisted file distribution: Single and multi-class problems," *submitted*, August 2006.

[9] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *Proc. of IEEE INFOCOM '06*, Barcelona, Spain, 2006.

[10] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. A. Felber, A. A. Hamra, and L. Garces-Erice, "Dissecting bittorrent: Five months in a torrent's lifetime," in *Proc. of PAM '04*, Antibes Juan-les-Pins, France, 2004.

[11] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough," in *Proc. of ACM IMC '06*, Rio de Janeriro, Brazil, 2006.

[12] L. Massoulie, A. Twigg, C. Gkantsidis, and P. Rodriguez, "Randomized decentralized broadcasting algorithms," in *Proc. of IEEE INFOCOM '07*, Anchorage, AK, 2007.

[13] J. Edmonds, "Edge-disjoint branchings," in *Proc. of the 9th Courant Computer Science Symposium on Combinatorial Algorithms, Algorithmics Press*, 1972, pp. 91–96.

[14] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *Proc. of IEEE INFOCOM '05*, Miami, FL, 2005.

[15] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang, "Measurements, analysis, and modeling of bittorrent-like systems." in *Proc. of ACM IMC'05*, Berkeley, CA, 2005.

[16] Y. Tian, D. Wu, and K.-W. Ng, "Analyzing multiple file downloading in bittorrent," in *Proc. of ICPP '06*, Washington, DC, 2006.

[17] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: high-bandwidth multicast in cooperative environments," in *Proc. of ACM SOSP '03*, Bolton Landing, NY, 2003.

[18] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *Proc. of ACM SOSP '03*, Bolton Landing, NY, USA, 2003.

[19] H. Zhang, G. Neglia, D. Towsley, and G. L. Presti, "On unstructured file sharing networks," in *Proc. of IEEE INFOCOM '07*, Anchorage, AK, 2007.

[20] M. Jain and C. Dovrolis, "End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput," *IEEE/ACM Trans. Netw.*, vol. 11, no. 4, pp. 537–549, 2003.

[21] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman, "A survey of gossiping and broadcasting in communication networks," *Networks*, vol. 18, pp. 319–349, 1988.

[22] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," in *Proc. of PAM'03*, La Jolla, CA, 2003.

[23] G. Smaragdakis, N. Laoutaris, P. Michiardi, A. Bestavros, J. Byers, and M. Roussopoulos, "Swarming on optimized graphs for n-way broadcast," Boston University, Tech. Rep. BUCS-TR-2007-009, July 2 2007.

[24] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach using BSP and MPI*. Oxford University Press, 2004.

[25] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, 2004.

[26] A. Singh, M. Castro, P. Druschel, and A. Rowstron, "Defending against eclipse attacks on overlay networks," in *Proc. of ACM SIGOPS European Workshop*, Leuven, Belgium, 2004.

[27] N. Laoutaris, G. Smaragdakis, A. Bestavros, and I. Stavrakakis, "Mistreatment in distributed caching groups: Causes and implications," in *Proc. of IEEE INFOCOM '06*, Barcelona, Spain, 2006.

APPENDIX A
NP-HARDNESS OF MAXIMIZING THE MINIMUM MAX-FLOW

Consider a node s that wants to select a set of neighbors σ from a network composed of m nodes $v_i \in V$, n nodes $u_j \in U$, and a single node t, so as to maximize its broadcast bandwidth defined to be its minimum max-flow to any destination, i.e., $\Phi(s,\sigma) = \min_{x \in (V \cup U \cup \{t\})} MF(s,x,\sigma)$, where $MF(s,x,\sigma)$ denotes the max-flow from s to x under strategy σ. Node s can use $k < m$ links whose bandwidth is b_1 if the other end-point belongs to V, and $\epsilon \approx 0$ in any other case, implying that an optimal strategy σ for s must satisfy $\sigma \subset V, |\sigma| = k$. Each node v_i has directed links of bandwidth b_2 to a subset U_i of the nodes of U. Each node u_j has a link of bandwidth b_3 to t. Node t has links of bandwidth b_1 to all nodes of V and U (see Fig. 5 for an illustration). Link bandwidths obey:

$$b_1 \gg b_2 \gg b_3 \tag{1}$$

Let $\phi(s,X,\sigma) = \min_{x \in X} MF(s,x,\sigma)$ denote s's minimum max-flow to any node in the set X. Combining $k < m$ and (1), we get that under any σ, at least one node of V will get s's flow only indirectly through t, i.e.:

$$\phi(s,V,\sigma) = MF(s,t,\sigma) \tag{2}$$

The max-flow from s to u_j is equal to the max-flow from s to t, plus b_2 for each connected path $s \to v_i \to u_j$ under σ, minus the amount of flow that crosses the link from u_j to t in a max-flow from s to t under σ. Since this flow on the (u_j,t) link is either 0, or $b_3 < b_2$ when there's at least one connected path $s \to v_i \to u_j$ in σ, we get $MF(s,u_j,\sigma) \geq MF(s,t,\sigma)$, $\forall u_j \in U$, or equivalently:

$$\phi(s,U,\sigma) \geq MF(s,t,\sigma) \tag{3}$$

The max-flow to node t is:

$$MF(s,t) = b_3 \cdot paths(s,V,U) \tag{4}$$

where $paths(s,V,U)$ is the number of connected paths $s \to v_i \to u_j$ that do not share (v_i,u_j) edges, or equivalently the number of nodes u_j that carry a non-zero flow in a max-flow from s to t. Equations (2), (3) suggest that the maximization of the broadcast bandwidth calls for the maximization of $MF(s,t)$, which in view of (4), is achieved through the maximization of $paths(s,V,U)$. Maximizing $paths(s,V,U)$ requires choosing k subsets V_i so as to maximize the cardinality of their union. A straight-forward reduction from set-cover can be used to show that max $paths(s,V,U)$ is an NP-hard problem (see Appendix C). Therefore, maximizing the broadcast bandwidth is also NP-hard as it implies a solution to max $paths(s,V,U)$.

APPENDIX B
NP-HARDNESS OF MAXIMIZING THE SUM OF MAX-FLOWS

Consider a node s that wants to connect to a network composed of m nodes $v_i \in V$, n nodes $u_j \in U$, and h nodes $w_l \in W$, where h is a function of the out-degrees of the v_i's as will be explained shortly, so as to maximize the sum of its max-flows to all nodes in the union of V, U, W. Node s can use $k < m$ links whose bandwidth is 1 if the other end-point belongs to V, and $\epsilon \approx 0$ in any other case, implying than an optimal strategy σ for s must satisfy $\sigma \subset V, |\sigma| = k$. Each node v_i has directed links of unit bandwidth to a subset U_i of the nodes of U. Each node u_j has a link of unit bandwidth to each one of the nodes of W (see Fig. 6 for an illustration). The cardinality of W is equal to the highest out-degree of any node in V, i.e., $h = \max_{1 \leq i \leq m} |U_i|$.

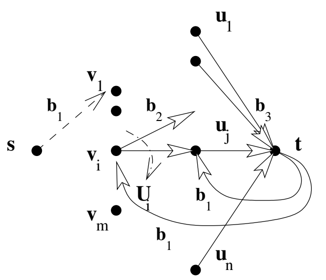

Fig. 5. Reduction from MAX-UNIQUES(k) to Max-Min.

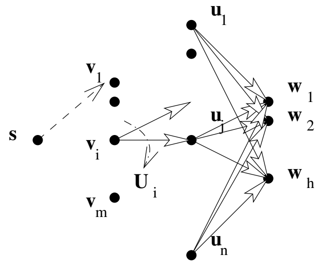

Fig. 6. Reduction from MAX-UNIQUES(k) to Max-Sum.

Define $\psi(s,X,\sigma) = \sum_{x \in X} MF(s,x,\sigma)$ where $MF(s,x,\sigma)$ denotes the max-flow from s to x under strategy σ. Node s wants to select a strategy σ that maximizes $\Psi(s,\sigma) = \psi(s,V,\sigma) + \psi(s,U,\sigma) + \psi(s,W,\sigma)$ across all possible strategies. We will show that such an optimal strategy has to maximize the number of nodes in U to which there exists a connected path $s \to v_i \to u_j$.

Notice that $MF(s,v_i,\sigma) = 1$ iff $v_i \in \sigma$ and 0 otherwise, and thus $\psi(s,V,\sigma) = k$ independently of the particular strategy σ chosen. Therefore, we only need to care to maximize $\psi(s,U,\sigma)+\psi(s,W,\sigma)$. If s chooses to connect to v_i, meaning $v_i \in \sigma$, the contribution to $\psi(s,U,\sigma)$ will be $|U_i|$, because each outgoing link of v_i increases by 1 every max-flow from s to a node $u \in U_i$. The contribution to $\psi(s,W,\sigma)$ will be h for each node $u \in U$ that is reachable from s if v_i is included in σ but becomes unreachable if it is taken out ("connecting" u increases all max-flows from s to nodes $w \in W$ by 1). Therefore if by switching $v_i \in \sigma$ with $v_{i'} \notin \sigma$ we get a strategy σ' which has a higher number of nodes of U reachable from s, then we should perform the switch because $\Psi(s,\sigma') > \Psi(s,\sigma)$. To see that, notice that the switch can hurt $\psi(s,U,\sigma)$ by at most $h-1$, if v_i has the highest degree and $v_{i'}$ has degree 1 (it must have at least 1 to be increasing the number of unique u's reached), whereas it benefits $\psi(s,W,\sigma)$ by at least h as it increases the number of nodes of U reachable from s. The above argument implies that an optimal σ must maximize the number of unique nodes of U reachable from s. Therefore, an optimal solution to maximizing the sum of max-flows for s implies an optimal solution to the NP-hard problem MAX-UNIQUES(k) of Appendix C. Therefore, max sum max-flows is also an NP-hard problem.

APPENDIX C
NP-HARDNESS OF MAXIMIZING THE NUMBER OF UNIQUES

Let MAX-UNIQUES(k) be an optimization problem in which one has to select k subsets U_i, $1 \leq i \leq m$ of a set U with n elements so as to maximize the cardinality of the union $U(k) = \bigcup_{i \in choice} U_i$. Let UNIQUES($k$) be the corresponding decision problem in which one asks whether there is a *choice* leading to $|U(k)| = l$. UNIQUES(k) is clearly NP-complete because for $l = n$ a solution to UNIQUES(k) would imply a solution to SET-COVER. Therefore, MAX-UNIQUES(k) is NP-hard.