oEvolve: Towards Evolutionary Overlay Topologies for High Bandwidth Data Dissemination

Ying Zhu, Jiang Guo, and Baochun Li, *Member, IEEE*

***Abstract*— In this paper, we consider the problem of data dissemination from a source to multiple receivers over application-layer overlay networks, and seek to significantly improve end-to-end throughput of data dissemination sessions by constructing topologies of high quality. We propose *oEvolve,* a distributed algorithm that uses the strategy of *progressively and adaptively evolving* the overlay topology over time towards high-quality topologies, especially with respect to end-to-end throughput of data dissemination. To validate the effectiveness and efficiency of *oEvolve,* we present a fully distributed real-world *oEvolve* implementation over *PlanetLab,* a global-scale wide-area overlay network testbed. Our implementation consists of a framework of components that involves a high-performance data forwarding engine and a centralized performance monitoring facility.**

***Index Terms*— Application-layer overlay networks, application-layer multicast.**

I. Introduction

Application-layer data dissemination on overlay networks has received enormous interest from the research community in the past few years, and has proven to be an attractive and viable alternative to IP multicast. The high hopes originally held for IP multicast — namely, its wide deployment in the Internet — have thus far failed to be realized, despite that it has been implemented and is available on most routers. IP multicast has serious inherent problems that are obviated in application-layer overlay networks. Transmission of data between end-systems does not require infrastructure change, as is the case for IP multicast. Scalability and complexity are a problem in IP multicast because a router must keep group state information for many groups, whereas an end-system only needs to maintain group state for the group(s) to which it belongs. Data transmission by unicast also means that we can exploit all existing security, flow control and reliable delivery mechanisms that are readily available and mature. Finally, the application layer offers unprecedented flexibility and freedom to design algorithms that incorporate a variety of Quality-of-Service considerations, including bandwidth, latency, and robustness.

Instead of working on the IP-layer network consisting of routers and physical network links, data dissemination is carried out on the application-layer overlay network, which comprises end-systems as nodes and unicast connections as links. It is assumed that this higher layer overlay network resides on top of a densely connected IP network. A link in the overlay network is a unicast connection between two end-systems; it is a virtual link corresponding to a path in the physical network that is invisible or unknown on the application layer. Data is transmitted from source to receivers by traversing on these virtual links between end-systems. The problem thus becomes: How should the routing be done by the end-systems to accomplish optimal or near-optimal data dissemination according to a particular metric or set of metrics, such as delay and bandwidth?

Application-layer overlay networks have two unique properties that both provide flexibility and introduce impediments when designing overlay algorithms. First, *topological variability*. The topologies of overlay networks have the potential to be *variable* in nature, since *every pair* of overlay nodes (end-systems) can establish an overlay (virtual) connection via the underlying transport protocol such as TCP. Further, these overlay nodes are mostly transient when participating in the overlay network, influenced by its own dynamics of joining, leaving and failures. Second, *link correlation*. We define this term precisely in Sec. II. In short, overlay links that share one or more physical links are naturally correlated. Since these correlations are not known by the application-layer algorithms, an overlay link chosen for its high bandwidth may not yield the expected throughput due to a hidden link correlation — a physical link shared with other overlay links, unknowingly. The properties of topological variability and link correlation are not hard to observe, but have rarely been explicitly taken into consideration when designing new algorithms in previous literature.

The problem we study in this paper is that of disseminating large volumes of data from a data source to multiple receivers. Such data may be either elastic or multimedia traffic. Our objective is to design an algorithm to achieve high *end-to-end bandwidth* from the source to each receiver. We present *oEvolve*, a distributed overlay algorithm that seeks to maximize bandwidth by constructing an overlay data dissemination topology that progressively *evolves* and *adapts*, based on on-the-fly measurement of performance metrics. The topology begins from a tree spanning all receivers; thereafter, *oEvolve* commences sending data on the current topology *while* continually growing the topology by adding successive trees spanning only those receivers with sufficient remaining bandwidth. The topology is variable and evolves based on inferred knowledge of the network, by continuous observation of performance in the current topology and technique of estimating residual last-mile bandwidth.

We have designed and completed a real-world implementation of *oEvolve* that consists of several non-trivial components at the application layer. In order to implement the *oEvolve*

The authors are affiliated with the Department of Electrical and Computer Engineering, University of Toronto. Their email addresses are {*yz,jguo,bli*}*@eecg.toronto.edu*.

algorithm for realistic delivery of data, we have designed and implemented a high-performance message forwarding engine entirely in the application layer, as well as a centralized monitoring facility for illustration and debugging purposes. In order to evaluate the performance of *oEvolve*, we deploy the *oEvolve* framework in PlanetLab [1], a wide-area overlay network testbed that spans over 100 nodes[1].

The contribution of this paper lies in three constituents. The first is the explicit mechanisms used in the algorithm which estimate last-mile bandwidth, and detect hidden bottlenecks and changing conditions in the network that cause declination in bandwidth. The second is the variability and evolutionary nature of the topology construction. Lastly, our implementation in a real-world wide-area network shows that the algorithm achieves our objective of high bandwidth overlay data dissemination.

The remainder of this paper is organized as follows. In Sec. II, we present salient properties of overlay data dissemination and provide theoretical insights. *oEvolve*, the main algorithm, is presented in Sec. III and IV, along with an analysis of its complexity and applicability. We proceed to evaluate our *oEvolve* implementation in *PlanetLab*. The design and implementation of underlying components of *oEvolve* are presented in Sec. V, and evaluation results of *oEvolve* are presented in Sec. VI. Finally, Sec. VII evaluates our proposal in the context of related work, and Sec. VIII concludes the paper.

II. Insights on High Bandwidth Overlay Data Dissemination

In its general form, a data dissemination session may be modeled by a graph representing its overlay topology, where the vertices are overlay nodes, and the edges are overlay connections between the nodes. In the Internet, it is generally the case that data packets between end-systems in a unicast connection follow the same path on the IP layer, due to the slow (relative to data session time) changing nature of IP routing. Hence, *we assume that each virtual overlay link maps to a unique path in the underlying IP network*.

We perceive and elaborate on three essential inherent properties of application-layer overlay networks: *link correlation*, *topological variability*, and *heterogeneous bandwidth availability* of overlay nodes.

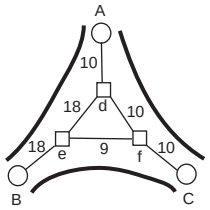
A. Link Correlation

The single most vexing difficulty in constructing any kind of high quality overlay topologies is the lack of knowledge of the underlying physical network. Virtual links in the overlay network may and often share common physical links, thus becoming what we call *correlated*. When the mapping from virtual to physical is unknown, the capacities of shared links induce unforeseen constraints on the flow rates (or bandwidth) on the overlay links. More precisely, when we say two overlay links are correlated, we mean the *flows* on these links are correlated; the definition is given below.

[1]At the time of our experiments and deployment on PlanetLab in October, 2003, there were approximately 100 active participating nodes in PlanetLab.

Definition 1 (link correlation) The flows $f(P_1)$ and $f(P_2)$ on virtual overlay links P_1 and P_2, respectively, are *correlated*, if P_1 and P_2 map to paths in the physical network that share a common edge e. $f(P_1)$ and $f(P_2)$ are correlated by the linear constraint: $f(P_1) + f(P_2) \leq c(e)$, where $c(e)$ denotes the capacity of e.

The definition naturally extends to multiple overlay links of arbitrary number, e.g., if P_1, P_2, P_3 share a common physical edge e, $f(P_1) + f(P_2) + f(P_3) \leq c(e)$. We use a simple example to illustrate how hidden link correlation may substantially reduce achievable bandwidth. Consider the network in Fig. 1: *A, B, C* are overlay nodes, *d, e, f* are IP routers, and the edges are physical network links. The unique mapping of the overlay links to the paths in the physical network is as follows. AB maps to (A, d, e, B), AC to (A, d, f, C), BC to (B, e, f, C). The physical link bandwidths are as labeled in Fig. 1. The measured bandwidths of overlay links AB, AC, BC are $10, 10, 9$, respectively. Suppose A is the source, and B, C are the receivers. The multicast tree with the highest receiver throughput, based *only* on knowledge of the overlay link bandwidths, would consist of $A \rightarrow B$ and $A \rightarrow C$. Due to the unknown sharing of Ad by AB and AC, the throughput for both B and C is only 5. However, given knowledge of the mapping of the overlay to the physical network, it is easy to see that the multicast tree with the highest throughput for B, C consists of the edges $A \rightarrow B$ and $B \rightarrow C$, yielding a throughput of 9 for both B and C.



An overlay network laid on top of physical network. Nodes A, B, are overlay nodes; nodes d, e, f are IP routers. Physical links are shown in thin lines; overlay links are shown in bold lines. All physical links are labeled with their capacity.

Fig. 1. An example of link correlation in an overlay network.

Complete information of the overlay-to-physical mapping leads to optimal-bandwidth overlay topologies. Unfortunately, obtaining complete or even non-trivial knowledge of such mappings usually involves much complexity and incurs high network costs. Existing research results of *topology discovery* [2] have estimated that the number of IP layer backbone routers may be in the proximity of $50,000$. It is therefore not feasible to obtain the complete mapping information or to explicitly compute the optimum based on such information. Low complexity in both communication and computation being a concern, we must find lightweight methods to handle link correlations.

B. Topological variability

The topology of overlay networks is *variable* in nature for two reasons. First, an overlay node is able to establish an overlay connection with any other overlay node (whose IP address is known). The topology of any overlay network can potentially be a complete graph. Thus, overlay networks can be constructed to be any topology by dynamically adding and

removing overlay links. Second, the presence of a particular overlay node is *transient* in nature since it may accidentally fail, or voluntarily leave and join the overlay topology at any time.

The variability of overlay topologies has further declared the infeasibility of using any knowledge of the underlying physical topology to optimize the end-to-end bandwidth of a data dissemination session. Even if such knowledge may be obtained with costs, it is highly likely that the topology has already changed to a different configuration by the time such knowledge becomes available for computations. Such variability makes it hard to justify high costs of obtaining knowledge of the IP topology.

C. Heterogeneous bandwidth availability

Heterogeneity in last-mile bandwidths of different access technologies (e.g., DSL or cable modem access) is clear evidence that overlay nodes vary significantly in their bandwidth availability. However, in an overlay data dissemination topology, due to limited buffering capabilities at overlay nodes, it is not trivial to keep track of the different sets of data received by the nodes with different receiving rates. In a tree topology, due to the constraints of per-node buffer capacities, the effective end-to-end throughput from the source to all the receiving nodes converges to the same value, which is that of the node with the lowest available last-mile bandwidth. Moreover, even if the tree had different receiver throughput values, nodes are still restricted to the receiving throughput of their upstream nodes.

Due to the high bandwidth of many end-systems and the Internet backbone, it is almost certain that there is bandwidth available in the network aside from that used by a single data dissemination tree. Our aim is to utilize the residual bandwidth to increase throughput for at least some of the nodes that have more bandwidth available to them. With established mechanisms such as source coding, heterogeneous receiver throughput is entirely feasible.

D. Last-mile bandwidth heuristic

We make the observation that last-mile links are most likely bottleneck links for paths between overlay nodes. Even though the Internet backbones use optical fiber extensively for superior transmission speeds, such high-bandwidth technology is still unavailable to users in the last-mile links, [3], [4], [5]. In [4], IBM research reports data (courtesy of Internet Society) that shows a factor of 8 *per year* of increase in World Wide Web demand from users in the 1990's. Clearly, the capacity of links must increase at more than this rate to provide higher available bandwidth to users. This level of communication infrastructure overhaul has not taken place for the vast majority of last-mile links; nor is it feasible for it to occur in the near future. Hence, we design a lightweight algorithm that does not attempt to discover full knowledge of the underlying IP topologies, but only localized knowledge of flow constraints on the *last-mile links*.

E. Max-min Fairness

Because of our aim of heterogeneous receiver throughput, the concept of *fairness* must be considered. We adopt the principle behind *max-min fairness*, a well-known and widely-recognized definition of fairness, [6].

The general idea of max-min fairness is to allocate flows to receivers in such a way as to maximize the allocation of each receiver i, subject to the constraint that any increase in i's allocation does not lead to a decrease in another receiver's allocation that is already as small as or smaller than i's. In other words, the receiver with the lowest available bandwidth is given the greatest possible allocation, and the same allocation is given to all the other receivers. From the remaining flows, the greatest possible allocation is given to the receiver with the lowest available bandwidth among the remaining receivers, and the same allocation given equally to the other receivers. This continues until all the flows are allocated.

Our design of *oEvolve* is based on the principle of max-min fairness. We start by constructing an initial tree of high quality spanning the source and all the receivers, on which data dissemination commences. When the flows have stabilized, the throughput for all the receivers converges to the same value — the bottleneck bandwidth. It is very likely that such bottleneck bandwidth is limited by the minimum last-mile bandwidth, since it is highly probable that some receivers have limited available last-mile bandwidth, especially as the number of participating nodes scales higher. The next step is to construct another tree using the residual available bandwidth for those receivers that have available last-mile bandwidth remaining. The topology grows as more trees are added until there is insufficient bandwidth for another source-rooted tree.

III. oEvolve: Distributed Algorithm

The gist of the *oEvolve* algorithm is to first construct a basic multicast tree including all receivers, rooted at the source. After the initial tree has been constructed, the data dissemination topology is further *evolved* to add new trees that include different sets of receivers, until the residual bandwidth of the receivers are saturated. The procedure for forming the second and subsequent trees is different from that for the first tree. It involves steps in which if a receiver detects a degradation in its throughput when it joins the new tree, then the receiver withdraws from the new tree.

As the topology evolves to include new trees, data continues to be disseminated on existing and new trees simultaneously. Disparate sets of data streams are disseminated on different trees in the topology. All the trees in the topology are dynamically constructed — *as well as destructed* — over time: the construction occurs when there is available residual bandwidth, and the destruction occurs when all nodes in one particular tree have resigned from the tree, in which case the source automatically terminates disseminating on the destructed tree.

We now formally present the details of the *oEvolve* algorithm.

A. Node state maintenance and dissemination

In *oEvolve*, every overlay node stores a *group list*, which is a list of the addresses of all the nodes it knows about in the *data dissemination group*, which is defined as the set of all receivers that are currently participating in the data dissemination session. With a locally maintained group list, an overlay node periodically and randomly selects a subset of nodes from its group list, exchanges local group lists with them, and updates accordingly upon receiving their group lists.

The *oEvolve* algorithm makes use of two types of metrics that each overlay node u stores:

1. *Link metrics:* The link from u to v, (u, v), on the overlay network has associated with it two metrics, $\beta(u, v)$ and $\lambda(u, v)$, where β is link bandwidth and λ is link delay.
2. *Node metrics:* Once the first multicast tree is constructed, the topology is initialized. As it evolves to include new trees, for a node u, its *node degree* $\delta(u)$ is defined as the degree of u in the current tree being constructed. In addition, its *node residual bandwidth* $\gamma(u)$ is defined to be the remaining last-mile bandwidth, with the presence of existing data traffic on the current topology.

The total degree of a node in a data dissemination topology is the number of its neighbors in the topology serving live traffic, including all its upstream and downstream nodes.

B. Constructing the initial oEvolve topology

The initial *oEvolve* topology is constructed from a connected graph of all the overlay receivers in the group and the source. Such a graph is referred to as the *basis graph*. To maintain the basis graph, in addition to the group list, each overlay node u also keeps a *neighbor list*, which contains a list of adjacent nodes in the basis graph. Nodes in the group list are randomly probed by u to measure the bandwidth of overlay links; those with the highest-bandwidth links to u are selected to be neighbors. The selection process continues until the maximum number of neighbors is reached.

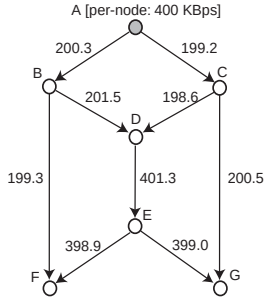
Then, the initial *oEvolve* topology is constructed on the basis graph, using the distributed all shortest-widest[2] paths algorithm based on distance vectors, proposed in [7]. With this algorithm, the widest path (highest end-to-end bandwidth) is selected; and if there is more than one widest path, the shortest (lowest end-to-end latency) is then selected. As such, when the construction completes, the initial data dissemination topology is a tree, on which data dissemination begins immediately.

C. Convergence to a uniform throughput: the "throttling effect"

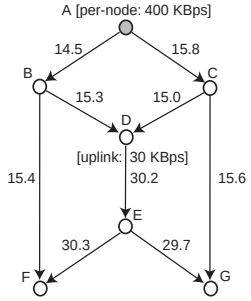
As data is disseminated from the source to all the receivers in the initial topology, we argue that the TCP flow control will ensure that the flow in the topology will *stabilize to one rate*. This is due to the limited buffering capabilities of overlay receivers. The uniform receiver rate after convergence is the lowest last-mile bandwidth or the capacity of the bottleneck link in the network. We refer to this conjecture as the "throttling effect" in overlay networks.

We show our experimental results verifying the above conjecture using a wide-area software infrastructure that we have developed for real-world experiments (implementation details deferred to Sec. V). In the topology we constructed, as shown in Fig. 2(a), the source A sends back-to-back traffic as rapidly as possible to the receivers. Identical copies of messages are sent to downstream nodes. When there exist multiple upstream nodes, no merging is performed. Buffers of all nodes were set to size of 5 messages and available bandwidth of node A was specified to be 400 KBps.

We observed that the throughput values on all the links have converged to correct values, as shown in Fig. 2(a). Then we proceeded to decrease the uplink available bandwidth of node D to only 30 KBps. In a few seconds, the throughput values of all the links except EF and EG have converged to those shown in Fig. 2(b). At node D, both incoming links have converged to 15 KBps due to the flow conservation property (no merging performed); while at node B, since BD is currently the bottleneck and messages have to be copied to both downstreams, both AB and BF are therefore throttled to the same throughput as BD.



(a) The traffic topology. A is the application data source, with a per-node total available bandwidth of 400 KBps, and copies are made when forwarding to multiple downstream nodes. The measured throughput values are marked at the edges, in KBytes per second.



(b) When the uplink available bandwidth of D is updated to 30 KBps, throughput of all links decrease to 15 KBps, except EF and EG which converge to 30 KBps. All the links are affected, rather than downstreams only, due to the back pressure from full buffers.

Fig. 2. Realistic testbed experiments that verify the "throttling effect" in overlay topologies. All flows in the topology stabilize to the same as or smaller than the bottleneck bandwidth capacity.

D. Estimating last-mile residual bandwidth

All flows into a receiver are correlated because they share its last-mile link. The linear constraint for flows on a last-mile link l is simply that the sum of all the flows on l be less than or equal to the bandwidth of l. With live data dissemination on the initial topology, one flow already exists on each l. Further allocation of flows into and out of each receiver must be constrained by the *residual bandwidth* of the overlay receiver.

There exist a variety of ways to measure the residual bandwidth on each of the receivers. We briefly describe one such mechanism from previous work (Jain *et al.* [8]). In this mechanism, we take advantage of the observation that one-way delays of a periodic packet stream may show an increasing

[2]Following convention, we henceforth use the terms *width* and *length* as informal ways to refer to *bandwidth* and *latency*, respectively. In particular, a *shortest widest* link refers to that with the smallest latency of all the links with the highest bandwidth.

trend when the stream's rate is higher than the available bandwidth. This mechanism is shown to be accurate and does not cause any significant increases in network utilization.

Alternatively, we may also use active probes to measure the residual bandwidth of a node. The proposal is to ask a particular overlay node to send back-to-back traffic to all its neighbors as much as possible for a very short period of time, while the existing data session is ongoing on the existing topology. The period of time should be less than 10 seconds, which is sufficient for TCP to saturate the residual bandwidth. The measured total throughput is an estimate of the residual last-mile bandwidth. This mechanism injects new traffic into the network; however, the overall costs are small since it does not need to be activated frequently.

The actual measurements may be performed sequentially from the source to downstream nodes. The source first performs the active probing, and when it has completed its probing process, it notifies all its downstream nodes, who are able to start the same procedure simultaneously to estimate their residual bandwidth. Alternatively, the measurements may also be performed randomly, such that each node decides the timing of the probing process independently. Since these measurements are not activated frequently, the probability of probes interfering with each other is small. Even if they do interfere, the smaller estimates which will result in a more conservative *oEvolve* algorithm in evolving the topology, and can be remedied in the next round of probing.

E. oEvolve core algorithm: evolving the data dissemination topology

Once the initial *oEvolve* topology is constructed, *oEvolve* enters the *evolutionary phase*, which lasts till the end of the data dissemination session. The key problem is to construct new trees for receivers who have residual bandwidth. To minimize fluctuations of constructing and destructing trees in the topology, in the implementation of *oEvolve*, a receiver joins a new tree only if its residual bandwidth is at least on the same order as its current data throughput, i.e., the residual bandwidth is greater than $1/10$ of the existing aggregate data throughput.

Let the subset of receivers with sufficient residual bandwidth be referred to as *the non-saturated receiver set*. Each non-saturated receiver stores a list of all known non-saturated receivers (along with their residual bandwidths), and this list is propagated and updated similarly as the group list, by all group members. Eventually, all non-saturated receivers know of each other, essentially forming a *complete* graph, referred to as the *residual basis graph*, G_B. The link bandwidths in G_B are bandwidth resources that remain from that used for the live data transmission in the existing topology.

We present the core of the *oEvolve* algorithm that constructs a tree in G_B, referred to as the *residual tree*, to be added to the current data dissemination topology. For the nodes that are already in the residual tree, their node metrics are maintained in a particular set, denoted as T. Each element of T is of the form $(u, \beta(u), \lambda(u), \delta(u), \gamma(u))$, where β, λ are the path bandwidth and latency from the source s, respectively, δ is the degree of u in the current residual tree, and γ is the residual bandwidth of u.

For $u \neq s$, the node metrics are initialized to $\beta(u) = \beta(s,u), \lambda(u) = \lambda(s,u), \delta(u) = 0$. For s, initially, $\beta(s) = \infty, \lambda(s) = 0, \delta(s) = 0$. The set T is initialized to $\{s, \beta(s), \lambda(s), \delta(s), \gamma(s)\}$. Let v denote the newest overlay node that is added to the residual tree and added to T (e.g., initially, $v = s$). Once v joins the tree, v chooses an overlay node u that is in G_B but not in the current residual tree, using the following criterion:

$$u = \arg\min_x \ \{\lambda(v,x) \ : \ x \in \arg \max_{y \in G_B - T} \ \beta(v,y) \ \}. \quad (1)$$

That is, (v,u) is the shortest widest link in G_B, where u is not already in the tree.

Upon receiving the set T from v, u joins the residual tree by selecting w as its upstream node (parent) such that

$$\begin{aligned} w \ = \ & \arg\min_y \ \Big\{ \ \lambda(y,u) + \lambda(y) \ : \ y \in \arg\max_{x \in T} \ \Big\{ \\ & \min \ \Big\{\beta(x), \beta(x,u), \frac{\gamma(x)}{\delta(x)+1}, \beta(u)\Big\}\Big\}\Big\}. \end{aligned} \quad (2)$$

Essentially, the width of a path from s to u through x — for some x already in the residual tree — is the minimum of the following four bandwidth values:

- path bandwidth from s to x in the residual tree;
- link bandwidth (x,u) in G_B;
- share of residual bandwidth of x that u would get if u joins the tree by choosing x to be its upstream node;
- residual bandwidth of u.

Node u simply chooses a node w in the current residual tree to be its upstream node, such that the tree path from s to u is the shortest widest.

After u is added to the residual tree, its upstream node w updates its node degree and path metrics:

$\delta(w) \leftarrow \delta(w) + 1;$
$\beta(w) \leftarrow \min\{\beta(w), \gamma(w)/\delta(w)\}.$

If w's path bandwidth changes, it is also made known to its descendants in the tree, since their path bandwidths may change due to the update. In *oEvolve*, w's updates are sent to its children in the residual tree. If a node z receives an updated $\beta(w)$, z computes $\beta(z) \leftarrow \min\{\beta(z), \beta(w)\}$ and forwards $\beta(z)$ to its own downstreams. Similar updates are performed by u:

$\delta(u) \leftarrow 1;$
$\beta(u) \leftarrow \min\{\beta(w), \beta(w,u), \gamma(u)/\delta(u)\}.$

Now that u is the newest node to join the residual tree, u adds itself and its appropriate metrics to T, and the above procedure repeats until T contains all the nodes in G_B. At this point, data is disseminated on the new tree, the latest addition in the evolving topology.

The above process of constructing residual trees continues, until the residual basis graph G_B, formed after a tree is constructed, does not include the source (i.e., the source bandwidth is saturated); or the source is disconnected from the other nodes; or it is empty (i.e., all the receiver bandwidths are

saturated). In other words, the evolutionary phase is suspended under *any* of the following conditions:

(1) the source has no residual bandwidth;
(2) no overlay receiver nodes has residual bandwidth;
(3) the network cannot sustain additional data flows, i.e., one or more physical links have reached their capacity, and to add more data flow would cause network congestion.

Beyond our baseline algorithm, the third condition above is vitally important. In Sec. III-F, we present mechanisms in the algorithm to detect potential network congestion and to vary existing topologies to avoid congestion.

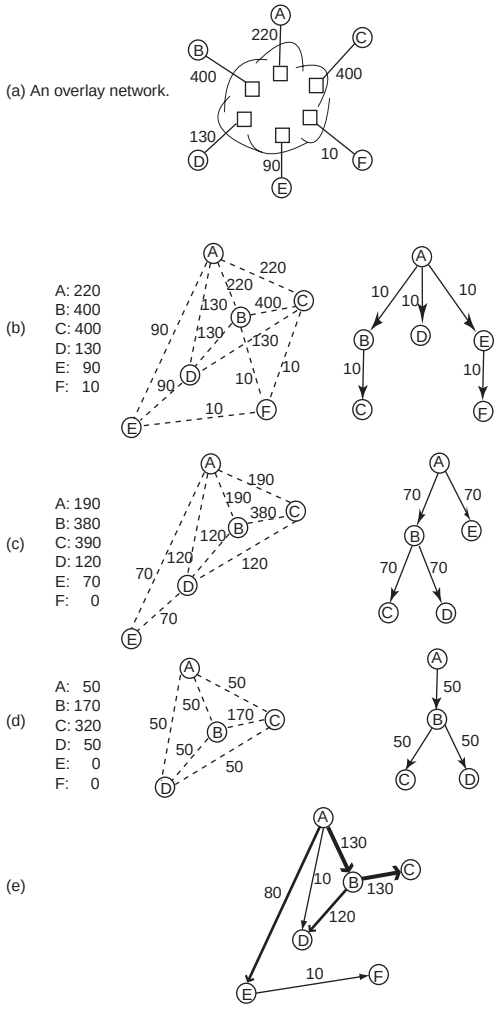


Fig. 3. The *oEvolve* core algorithm: an example. **(a)** An overlay network with last-mile bandwidths shown. **(b)-(d)** Successive iterations of the **while** loop. The leftmost column gives the residual bandwidth for each node at the start of each iteration. The left side of each pair of graphs shows the residual basis graphs (residual bandwidth labeled on edges) after a tree is built. The right side of each pair shows the new tree built based on the residual basis graph on the left. **(e)** Final data dissemination topology, combining all the trees. The bolder the line, the higher the number of tree links amalgamated. Each link is labeled with its aggregate throughput.

We now present an example, shown in Fig. 3, to illustrate the steps of the *oEvolve* core algorithm during the evolutionary phase. In Fig. 3(a), the last-mile bandwidths of the nodes in the overlay network are labeled, and the cloud represents the Internet. For simplicity of illustration, we make the (reasonable) assumption[3] that the Internet backbone has high enough capacity such that the last-mile bandwidths are the constraint on data flow into and out of nodes. It follows that the *bandwidth of an overlay link* between two nodes is the *minimum of their last-mile bandwidths*, as labeled in the initial basis graph — the graph on the left in Fig. 3(b). Successive iterations of the algorithm are shown in Fig. 3(b)-(d). The residual bandwidth of each node at the beginning of each iteration is shown in the leftmost column. Each pair of graphs give the residual basis graph on the left and the resulting residual tree on the right.

Suppose node *A* is the source and the other five nodes are all receivers. The initial topology produced by all shortest-widest paths algorithm — the tree on the right in Fig. 3(b) — yields a throughput of 10.

Now the evolutionary phase commences. Using the heuristics in the algorithm to estimate the last-mile bandwidth, *F* will deduce that it has depleted its last-mile bandwidth and refrain from joining further trees. Thus, the current residual basis graph is the graph on the left in Fig. 3(c).

The second tree, the graph on the right in Fig. 3(c), is constructed incrementally based on residual bandwidth. As an example, we look at node D when it joins the tree with $T = \{A, B, E\}$ (nodes already in the tree). D obtains T from one of its neighbors, and uses Eq. 2 to select a node from T to be the parent. The bandwidth of a path from A to D in the existing tree via B is 70, while via E it is 0 and directly from A it is 50. Similarly, C does not choose A, D or E to be its parent because the path from A to C via B has the highest bandwidth.

Again, heuristics are utilized by nodes to measure their residual bandwidth: E now detects that it has saturated its last-mile link and stops joining new trees. The residual basis graph after the first two trees is the graph on the left in Fig. 3(d). The third tree constructed is the graph on the right in Fig. 3(d), and the data dissemination topology (by amalgamating the three trees) with the aggregate heterogeneous throughput for each receiver is shown in Fig. 3(e). Henceforth, the topology evolves further in response to node dynamics and changing network conditions.

Note that in implementing the algorithm, it is desirable to maintain *one* connection between a pair of nodes, regardless of how many links there are between them from different trees. In our implementation, each tree has an identification number, which is stored in a field in the header of each message to facilitate routing.

F. Evolving the topology with the presence of varying network conditions

Generally, the dynamics of adverse overlay network conditions, such as significantly deteriorated throughput, are caused by bottleneck physical links reaching their capacity, since TCP will adjust its sending rates as bottleneck routers drop IP packets. A noticeable deterioration in receiving throughput is a

[3]The last-mile link is frequently the bottleneck, and even if that is not the case, the network congestion detection and avoidance component of *oEvolve* will prevent deterioration in performance.

clear indication of adverse network conditions. A receiver may passively, non-intrusively monitor its aggregate throughput and detect such adverse conditions.

oEvolve passively and continuously measures per-node aggregate TCP throughput. This modification to the basic *oEvolve* is made: *the source always aggressively starts to send data as soon as possible*. Such data dissemination continues as the tree is constructed. When u joins a tree, an active connection is established from its upstream to itself, and it starts to receive data from the source immediately. After an initial period of settling into a more stable flow rate of TCP (usually less than 10 seconds), u measures the new instantaneous aggregate throughput. When u joins a residual tree, two cases may arise:

> *Case 1*: If the new throughput is greater than the throughput before u joins the new tree, u remains in the tree;
> *Case 2*: If the throughput decreases, u immediately resigns from the new tree, and removes itself from the current residual basis graph.

If all of the nodes within a particular residual tree have resigned, the tree is destructed automatically. When a node u resigns from a tree, its downstream nodes would be forced to also resign. To address this problem, we propose to *locally repair* the residual tree in *oEvolve*: Upon resignation, u sends a notification message to all its immediate downstream nodes, encapsulating the information about its upstream (parent) v. As all downstream nodes receive the message, they proceed to contact v to reconnect into the residual tree.

G. Node joins and departures

When a node A *joins* the data session, it is given a list of group members that are currently participating in the session. Node A updates its group list by randomly contacting known members and obtaining their group lists. Node A also obtains the *non-saturated receiver set* by randomly contacting nodes until reaching one that stores this set. Receivers with high residual bandwidths are probed to measure the bandwidth of their unicast links to A. A *join* message is sent by A to those with both high link bandwidth and high residual bandwidth. A receiver, v, upon receiving the message, will return $\min\{\beta(v), \gamma(v)\}$, where $\beta(v)$ is the aggregate receiving throughput of v and $\gamma(v)$ is the residual bandwidth of v. That is, A collects from each v the highest possible throughput v can relay to A. The v with the highest value is selected to be its parent in all the trees to which v belongs. While A and the network both have sufficient residual bandwidth, A continues to find another upstream node and join more trees.

The procedure just described is *incremental* in nature. The incremental joining and the local repair after departure involve mainly locally attaching nodes to existing trees, and will potentially cause the dissemination topology to deteriorate in performance as more nodes join and leave. Therefore, after a certain number of nodes have joined or departed, the algorithm will *reconstruct* the topology from the beginning — the initial *oEvolve* topology and all subsequent residual trees.

A node sends notification to its neighbors when it *leaves* the group. The periodic exchanges of information between nodes ensure that eventually every node will be informed. If a node leaves without being able to notify others, then some node attempting to exchange lists with this node will discover its lack of response and send out notifications that this node is no longer in the group.

Previous measures to address the problem of a node leaving one residual tree is also effective when nodes depart from data dissemination sessions voluntarily. When a node leaves the session, it just needs to inform its immediate downstream nodes of information about its upstream nodes in *all* the residual trees in which it has participated, in order for the downstreams to repair the trees locally.

Finally, for the convenience of the reader, we present the *oEvolve* core algorithm in Table I and Table II.

TABLE I
SKELETON OF *oEvolve* CORE ALGORITHM

Let G_B be the current residual basis graph.
Let i denote the current new residual tree that node u is joining.
$i \leftarrow 2$; // initial topology is tree 1
Let f_u be the current aggregate throughput.
Estimate u's last-mile residual bandwidth $\gamma(u)$.
while $\gamma(u) > 1/d \cdot f_u$
  Update G_B.
  Let $T = \{a, \beta(a), \lambda(a), \delta(a), \gamma(a)\}$ be nodes already in tree i.
    // $\beta, \lambda =$ bandwidth & latency of path from s in tree i, resp.
    // $\delta =$ degree in tree i, $\gamma =$ residual last-mile bandwidth
  Upon receiving T from a node v:
    if $\max_{x \in T}\{\min\{\beta(x), \beta(x,u), \frac{\gamma(x)}{\delta(x)+1}, \beta(u)\}\} < 1/d \cdot f_u$
      // No residual bandwidth connecting s to u, do not join
      break;
    $w \leftarrow \arg\min_y \{ \lambda(y,u) + \lambda(y) : y \in \arg\max_{x \in T} \{\min \{\beta(x), \beta(x,u), \frac{\gamma(x)}{\delta(x)+1}, \beta(u)\} \} \}$;
    Send <*join tree i*> message to w.
    Receive w's updated node metrics from w.
    // Update u's own node metrics:
    $\delta(u) \leftarrow 1$;
    $\beta(u) \leftarrow \min\{\beta(w), \beta(w,u), \gamma(u)/\delta(u)\}$;
    $\lambda(u) \leftarrow \lambda(y,u) + \lambda(y)$;
    $T \leftarrow T \cup \{u, \beta(u), \lambda(u), \delta(u), \gamma(u)\}$;
  After data dissemination begins on tree i:
    Estimate residual bandwidth, $\gamma(u)$, and update it in T.
  $i \leftarrow i + 1$;
  Continual heuristics to detect deterioration in network conditions.

H. Analysis

We now discuss a few important properties of the algorithm. We analyze two types of complexity for our algorithm: *communication* and *time complexity*. The communication complexity is the number of control messages required for constructing the data dissemination topology; the time complexity is the time that the *oEvolve* algorithm takes to finish. Here, we consider *oEvolve* to be finished when all the residual trees have first completed their construction, assuming stable network conditions. After that, the data dissemination topology may likely vary as network conditions change and nodes withdraw or join various residual trees. However, this ongoing process should have complexity of the same order.

First, we note that the *oEvolve* algorithm terminates after a finite time. The algorithm uses distance vectors for routing

TABLE II
UPDATING AND PROPAGATING METRICS AFTER A NODE JOINS TREE i

Node w upon receiving <*join tree i*> message:
  // Update node degree
  $\delta(w) \leftarrow \delta(w) + 1$;
  if $\min\{\beta(w), \gamma(w)/\delta(w)\} < \beta(w)$
    // Update path bandwidth and notify children
    $\beta(w) \leftarrow \min\{\beta(w), \gamma(w)/\delta(w)\}$;
    Send <*update node bandwidth*, $\beta(w) >$ to children
  else do nothing

`Propagation`: Node z, downstream of w, upon receiving <*update node bandwidth*, $\beta(w) >$ from parent:
  if $\min\{\beta(z), \gamma(z)/\delta(z)\} < \beta(z)$
    // Update path bandwidth and notify children
    $\beta(z) \leftarrow \min\{\beta(z), \gamma(z)/\delta(z)\}$;
    Send <*update node bandwidth*, $\beta(z) >$ to children
  else do nothing

(among overlay nodes) and the convergence of distance-vector routing algorithm has been proven; the reader is referred to the proof in [9]. The algorithm always terminates because there is finite bandwidth, particularly at the source. Every time a tree is added, more residual bandwidth of the source is utilized; all residual bandwidth are eventually used in a finite time.

The initial stage of exchanging group lists is considered to have converged when every node has obtained a list of all nodes in the group. This is achieved by having nodes exchange their current group lists with some randomly selected nodes and update lists accordingly. It is easy to see that this procedure is exactly the same as distance vectors routing protocol, where group lists replace distance vectors. Therefore, its time complexity (or convergence time) is $O(n)$ and communication complexity is $O(n^2)$ (see [9]).

The traditional all-shortest paths algorithm based on distance vectors is known to have communication complexity of $O(n^2)$ and time complexity of $O(n)$ [10]. All shortest-widest paths algorithm is only different from all shortest paths algorithm in comparing widths first and then comparing the distances, and taking the minimum of widths in addition to summing the distances. Thus, the two algorithms clearly have the same communication and time complexity; hence the initial tree takes time $O(n)$ and control messages overhead $O(n^2)$.

We now analyze the complexity of the procedure for constructing subsequent residual trees. When a new residual tree is being constructed, for every node that joins, a constant number of messages are exchanged to obtain node metrics for comparison. Also, the updated node degree is propagated in the tree to all the descendants, which requires $O(n)$ messages and $O(\log n)$ time. Since at most n nodes are part of the new residual tree, the communication complexity is $O(n^2)$ and the time complexity is $O(n \log n)$.

It remains only to obtain an upper bound of the possible number of residual trees. Note that the algorithm enforces that the $(i+1)$-th tree has bandwidth to be at least $1/d$ of the sum of bandwidths of all previous i trees. The worst case is when each tree $i+1$ has bandwidth that is $1/d$ of aggregate bandwidth of the previous i trees. Let b_0 be the bandwidth of the first tree and k be the total number of trees. Then the total bandwidth of all k trees is $(\frac{d+1}{d})^k \cdot b_0$. Let l and u denote the lowest and highest possible last-mile bandwidth, respectively. Since b_0 is greater than l and $(\frac{d+1}{d})^k \cdot b_0$ is less than u, k is bounded by $\log_{\frac{d+1}{d}}(u/l)$.

It is interesting to note the relation between the upper bound of k and the lower bound of the new tree's bandwidth, $1/d$, which is a parameter set in the algorithm. With $1/d$ being $1/10$, k is $O(log_{\frac{11}{10}}(u/l))$. While if it is $1/2$, k is $O(log_{\frac{3}{2}}(u/l))$. In practical terms, we observe that the lower bound for node bandwidth is about 56 Kbps for a dial-up modem, and the upper bound is approximately 1 Gbps. The number of trees, k, is hence bounded by $\log_{\frac{d+1}{d}}(20,000)$, which is around 104 for $1/d = 1/10$ and 24 for $1/d = 1/2$. This presents a trade-off between throughput and efficiency. Using $1/2$ may result in lower throughput for some nodes because new trees giving a smaller fraction of throughput improvement would not be built. On the flip side, more trees will slow down the algorithm and add communication overhead.

Therefore, the total communication complexity for construction of the topology is $O(n^2)$ and the total time complexity is $O(n \log n)$, with a coefficient of $\log_{\frac{d+1}{d}}(u/l)$, which is independent of n, but dependent on d, l and u.

Thus far we have covered the complexity of the algorithm for constructing an entire data dissemination topology. The incremental joining procedure of a node involves transmission of a constant number of messages between the new node and potentially every other node, hence the communication complexity is $O(n)$, as is the time complexity. Local repairing upon a node departure requires in *worst case* $O(n)$ nodes to carry out an incremental join. Since the joins are in parallel, the time complexity is $O(n)$; however, the number of control messages is clearly $O(n^2)$.

I. Optimal solution

We briefly discuss the issue of a theoretical optimal solution. We have not theoretically derived the optimality of *oEvolve* due to the inherent difficulty of doing so for any such heuristics-driven, distributed network protocols, such as distance vector or link state. To the best of our knowledge, we are not aware of any result on the theoretical optimal solution that gives a flow assignment such that receiver throughput is maximized according to max-min fairness, for the *input* of an overlay network *and* the mapping from overlay to physical network. In [11], Garg *et al.* gave a linear programming formulation of maximizing *equal* receiver throughput in an overlay network with perfect knowledge of overlay-to-physical mapping and showed it to be polynomial-time solvable. However, this only gives a topology in which each receiver has the same throughput; we refer to this as *maximum uniform flow*.

The max-min fair optimal flow for heterogeneous receiver throughput is one in which for any receiver i with throughput r_i, r_i cannot be increased without decreasing some $r_j < r_i$. An algorithm can be conceived for such a *max-min flow*, using one for maximum uniform flow (MUF) that returns all possible

MUFs for the given graph and mapping. There may be more than one MUF with equal maximum flow value. The outline of the *max-min flow* algorithm is as follows. The network graph G and overlay-to-physical mapping M are the input. For each $F_{1,i} \in F_1$ — set of flows given by MUF(G, M), obtain the residual network graph R by subtracting $F_{1,i}$; let $\{f_j\}$ be the set of MUFs returned by MUF(R, M); and add $F_{1,i}+f_j$ to the set F_2. Keep only the maximum valued flows in F_2 (initially empty). For each $F_{2,i}$ in F_2, repeat as for F_1. Repeat this for F_k until MUF returns zero flow.

The algorithm given above is potentially exponential because there may be multiple MUFs at each step. We present an example with a small overlay network that compares *oEvolve* with the optimum, shown in Fig. 4. An overlay network with overlay nodes A, B, C, D, E is given in Fig. 4(a); A is the source and the rest are receivers. Fig. 4(b) shows the *oEolve* solution (note E is a child of D in the first tree because E did not know A at first). The trees are combined to yield the data dissemination topology, in which B, C, D, E have throughput $51, 10, 39, 49$, respectively. The two flow graphs shown in Fig. 4(c) amalgamate to yield the optimal max-min fair throughput for each receiver — B, C, D, E have throughput $48, 10, 48, 48$, respectively. The largest discrepancy is for D, where *oEvolve* is unfair to D in allocating 81% of the optimal bandwidth.

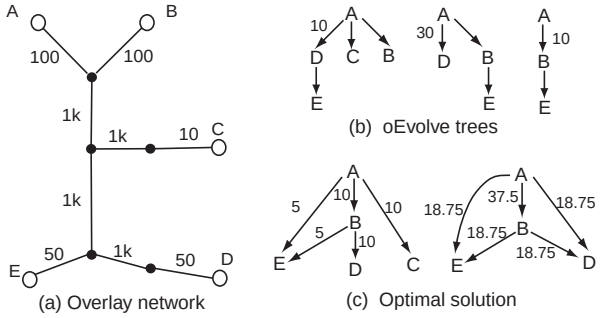


Fig. 4. An example comparing *oEvolve* with the optimum.

IV. Heterogeneous-throughput overlay data dissemination: Case Studies

We argue that heterogeneous-throughput data dissemination is most helpful in the cases of *multimedia streaming* and *content distribution*. For both cases, data can be encoded at the source and sent to the receivers. There are a number of source coding schemes that could be employed for the purpose of streaming media content (e.g., video streams) to receivers at different rates. One approach is layered coding [12], in which a signal is encoded into a number of layers that can be incrementally combined to yield ever increasing quality. This is completely natural to the topology generated by *oEvolve*, since any receiver has a path in each of the first i trees and hence receives the first i layers (i is the number of trees the receiver is in). Layered coding incurs only a slight performance penalty.

With content distribution applications such as file transfers, which must distribute the entire contents of a file to all the receivers, it is also feasible to use the *oEvolve* topology. Every receiver is in at least one *oEvolve* tree, and if it is not in all the current *oEvolve* trees, then some of its ancestors (in at least one of the trees it has joined) must participate in larger topologies (amalgamation of a higher number of trees). Receivers can recover the missing data from one or more of these upstream nodes in the data dissemination topology.

It should be noted that content distribution using *oEvolve* has two advantageous and desirable properties. First, *robustness*. In *oEvolve*, a new residual tree is only constructed if the residual bandwidth of the nodes is on the same order as the highest receiver aggregate throughput in the current topology. We may slightly revise this condition such that a new tree is constructed only if the residual bandwidth is greater than the current topology throughput. Hence, *oEvolve* trees have coarse grades of monotonically non-decreasing bandwidth and it is highly probable that the receivers that join all the current trees are not few. These receivers are the safeguard against node or link failures, since they receive all the contents and can in fact be viewed as multiple "sources", effectively mirroring the original source. There is not a single point of failure, and recovery of missing data has a higher level of parallelism and therefore is more efficient. Second, *ease of locating missing data*. Consider a receiver wishing to search for nodes with data that it does not have. It simply needs to send a query upstream in the last *oEvolve* tree that it has joined, and will be guaranteed to find an upstream node that has its missing data due to its participation in a larger topology. For better load balancing, it may choose to find another node, which can be efficiently accomplished by traversing upstream in a direct path in *oEvolve* trees. Furthermore, there is no complexity incurred by comparing disparate data sets and computing differences, since the data is already known to be exactly divided in segments corresponding to the bandwidth of *oEvolve* trees.

V. oEvolve: Implementation

We have designed and implemented *oEvolve* in the real-world *PlanetLab* wide-area overlay network testbed. For this purpose, we have designed and implemented two supporting components *from scratch*: (1) the *engine*, a high-performance application-layer message processing facility in UNIX, that is used to support live data dissemination sessions from the source to the receivers; (2) the *observer*, a centralized graphical monitoring facility in Windows, used to monitor and control various aspects of the *oEvolve* deployment over wide-area networks. The observer in action is shown in Fig. 5. We now present the salient capabilities of the engine and the observer as follows. Those readers who are interested in the design and implementation details of the engine or the observer are referred to [13].

Message processing. The engine is designed to smoothly support multiple groups of traffic belonging to different applications, or different data dissemination sessions. It also has the capability to concurrently process both application data and *oEvolve*-specific control messages. The design of the engine resembles an application-layer message switch, switching data from upstreams to downstreams.

Table III shows the skeleton of the core engine implementation. In this implementation, the `Algorithm` class is defined as a base class that the *oEvolve* implementation inherits. The `switch()` function switches messages from the receiver buffers to the sender buffers in a round-robin fashion.

TABLE III
DESIGN OF THE ENGINE THREAD

```
start the TCP server on the publicized port;
bootstrap from observer;
while not terminated
  if there are incoming messages on the port detected
  using non-blocking select()
    if the message is engine-related
      call Engine::process();
    else
      call Algorithm::process();
  call Engine::switch();
stop the TCP server.
```

Measurements of performance metrics. Important performance metrics such as per-link throughput and latency are measured by the engine. Both the *oEvolve* algorithm and the observer will be notified of the results of such measurements, as part of the ongoing status reports of the overlay node. For implementation detail on how the engine obtains measurements of these metrics, the reader is referred to [13].

Emulation of bandwidth availability. For verification purposes, we elect to perform preliminary tests of *oEvolve* under controlled environments, in which node characteristics are more predictable. The engine explicitly supports the emulation of bandwidth availability in three categories: (1) per-node bandwidth: the total incoming and outgoing bandwidth available; (2) per-link bandwidth: the bandwidth available on a certain point-to-point virtual link; and (3) per-node incoming and outgoing bandwidth: the engine is able to emulate asymmetric nodes (such as nodes on DSL or cable modem connections) featuring disparate outgoing and incoming bandwidth availability.

To use the engine, the *oEvolve* algorithm that we have implemented only needs to call *one* function of the engine: the *send* function. The *oEvolve* implementation is designed as a message handler, in the form of a *switch* statement on different types of messages. While processing each incoming message, internal states of *oEvolve* may be modified. The message handler resides in the `process()` function. The skeleton of the *oEvolve* implementation is shown in Table IV.

In such a skeleton, it is not necessary for *oEvolve* to handle all the known message types from the engine or the observer. If a message type is not handled in the *oEvolve* implementation, the default `process()` function provided by a base *Algorithm* class from the engine takes this responsibility.

VI. OEVOLVE: EVALUATION

In order to evaluate various aspects of the *oEvolve* algorithm and its wide-area implementation, we have conducted experiments with different sets of sites on PlanetLab, with the data source deployed at various sites. In this section,

TABLE IV
SKELETON OF THE *oEvolve* IMPLEMENTATION

```
oEvolve::process(Msg * m)
  switch (m -> type())
  case sDeploy: (from observer)
    deploy an application source;
  case request: (from observer)
    send oEvolve status updates to observer;
  case sTerminate: (from observer)
    terminate an application source;
  case BrokenSource: (from upstream)
    clear up internal states corresponding to the
    application source at upstream, since it has failed;
  case data: (from the engine)
    process, consume or forward the message using
    send(Msg * m, Node dest);
  case UpThroughput: (from the engine)
    record and process the throughput from an upstream;
  ... (process other engine or oEvolve-specific types)
  default: (use the default behavior from the base
  Algorithm class)
    Algorithm::process(m);
```

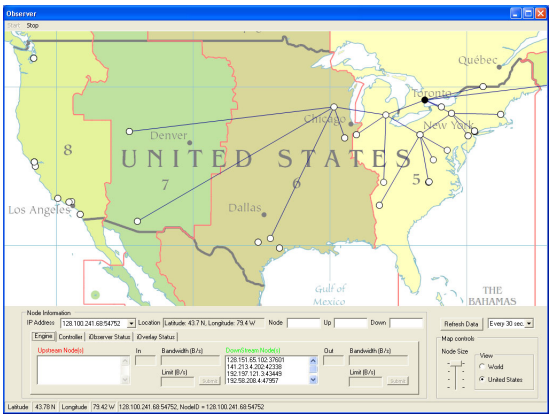


Fig. 5. *oEvolve* in action with an ongoing topology construction process, with 79 PlanetLab nodes deployed across the Internet.

we present a representative set of our experiments, from our deployment of *oEvolve* on 10, 50 and 79 sites on PlanetLab. For each experiment, we employ nodes with varying available bandwidth. Fig. 5 illustrates *oEvolve* in action in the observer, with 79 PlanetLab nodes across the Internet.

First, to illustrate the evolutionary nature of the topology, we show the data dissemination topology at four progressive stages in the *oEvolve* evolutionary phase for the deployment of 10 nodes, presented as four directed graphs in Fig. 6. The nodes are labeled with their IP addresses. The four graphs show the varying topology at each stage of its construction over time. It can clearly be seen from the figure that the source node has the highest degree at each step and in the final topology. This is explained by our having chosen a host that has a high-speed Internet connection to be the source for that experiment.

Next, we present results in an experiment with 50 nodes. The graphs in Fig. 7 represent the evolving topology at four progressive stages. As can be observed in Fig. 7, there are approximately three classes of nodes with various levels of

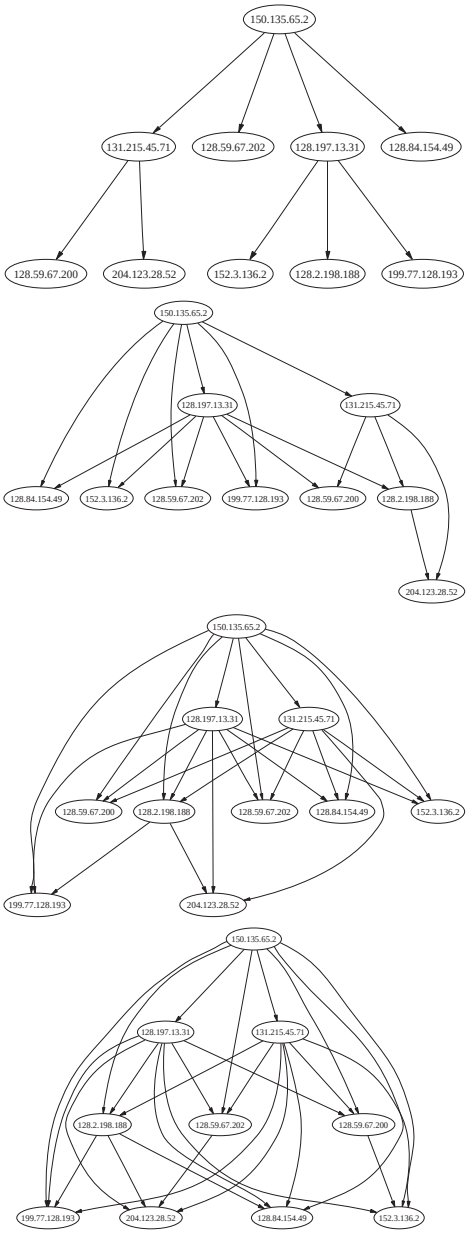


Fig. 6. Evolution of the data dissemination topology for 10 nodes on PlanetLab.

participation in the trees, indicating that the nodes have various levels of bandwidth. We executed `ping` on each of the 50 nodes and noted a variation of round-trip time — which gives a reasonable indication of bandwidth — that *corresponds* to the observed variation in the number of trees joined. This demonstrates that the number of trees to which a node belongs is proportional to its bandwidth, conforming to the aim of the algorithm. Once again, we selected a node with known high bandwidth to be the source, which is indeed reflected in the generated topology by the apparent high degree of the source node.

We proceed to illustrate, in Fig. 8, the end-to-end throughput for the nodes at each stage of the evolving topology. The bottommost curve is the throughput in the first tree, the second bottommost curve is the aggregate throughput in the first two trees, and so on. The corresponding cumulative distribution of throughput is presented in Fig. 9. A significant improvement in throughput clearly exists in the last topology compared with the initial topology (the first tree); even from the second last topology (second curve from the top) to the last topology (topmost curve), there is a noticeable increase in aggregate throughput for many nodes. In Fig. 8, heterogeneity of throughput is obvious: throughput varies greatly from node to node; those nodes at the top of the spikes of the topmost curve have the highest throughput. It is evident in both Fig. 8 and Fig. 9 that all the nodes gain a substantial improvement in throughput beyond the first tree. This is a sure indication that *frequently* there is residual bandwidth, both in the network and from the last-mile bandwidth of nodes, available to be utilized beyond a single multicast tree. We observe a noticeable increase in throughput even going from the second last topology to the last topology, it is a convincing indication that there is often residual available bandwidth after building multiple multicast trees, that may be further used to improve throughput for certain nodes. The slight fluctuation in throughput for nodes in the first tree, in Fig. 8, can be explained by changing network conditions and imprecision in throughput measurements.

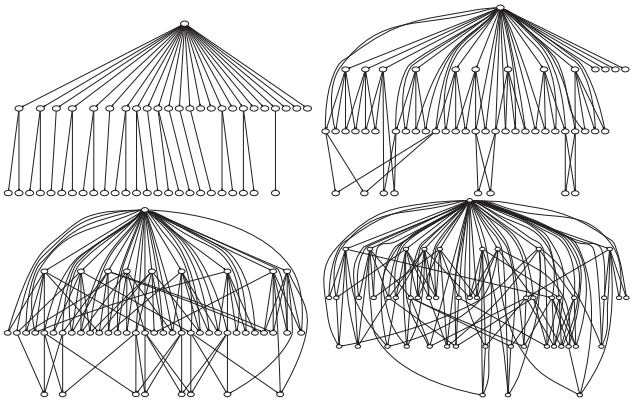


Fig. 7. Evolution of data dissemination topology for 50 nodes on PlanetLab.

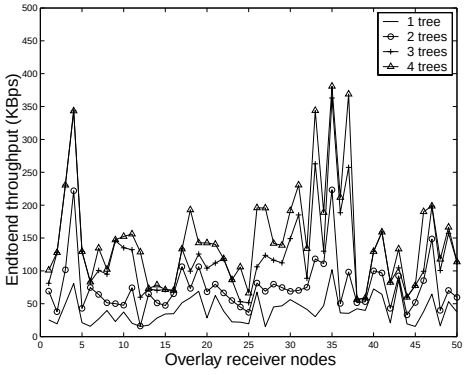


Fig. 8. End-to-end throughput for 50 node experiment for each stage of the evolving topology.

In the case of the 79-node experiment that we have conducted across the Internet, we present the constructed data dissemination topology in Fig. 10 (we choose not to show

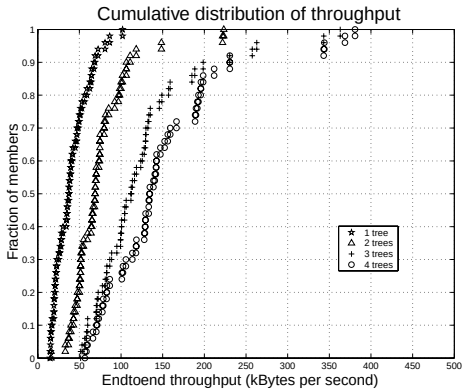


Fig. 9. Cumulative distribution of end-to-end throughput for 50 node experiment for each stage of the evolving topology.

the intermediate topologies due to space constraints). For this deployment, we did not choose a source node that has higher bandwidth than all the other nodes. Instead, the source is only one among a number of nodes with (approximately) equally high bandwidth. The topology in the graph verifies this: the source does not have a noticeably higher degree than all other nodes.

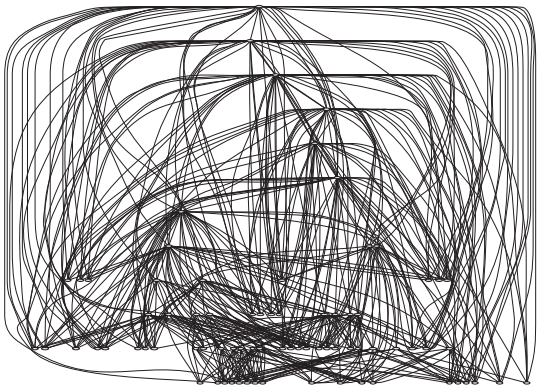


Fig. 10. Data dissemination topology in the 79-node experiment in PlanetLab, before the transitional event introducing cross traffic.

To illustrate the adaptive and evolutionary nature of *oEvolve* topologies when there are deteriorating network conditions due to external cross traffic, our 79-node experiment consists of a transitional event. As 79 *oEvolve* nodes are deployed on the Internet and the initial construction of the data dissemination topology has been completed (as in Fig. 10), we deploy another multicast tree of 40 nodes as competing cross traffic. The nodes in the competing tree constitute a subset of the original set of 79 nodes. We illustrate the aggregate end-to-end throughput before and after the transition for all the 79 nodes in Fig. 11 and Fig. 13, respectively. Two curves are plotted in each figure — one for the throughput in the data dissemination topology generated by *oEvolve*, and the other for the throughput in a static 79-node multicast tree constructed by all shortest widest paths on the basis graph. In both cases, the topology generated by *oEvolve* enjoys much better throughput performance than the multicast tree. Most of the nodes achieve significantly higher throughput than they do in a multicast tree. The heterogeneity of throughput for the nodes is obvious in both figures.

After the transitional event in our 79-node experiment, the nodes with low bandwidths are evident in Fig. 13. On average, the throughput in Fig. 11 is higher than in Fig. 13, and the latter figure also has more nodes with low throughput, approaching that in the multicast tree. This shows the effects of *oEvolve* with respect to withdrawing from residual trees when deteriorating network conditions are detected with passive measurements. However, these results have confirmed our previous theoretical analysis that the higher bandwidth nodes are not affected by the low bandwidth nodes that are constrained by the competing traffic. The higher bandwidth nodes achieve consistently high throughput even while some of the other nodes have very low throughput. We also note that even though many nodes achieve much higher throughput, they do not do this at the expense of lower bandwidth nodes, which still perform better than in a multicast tree. This demonstrates that *oEvolve* has achieved max-min fairness with respect to end-to-end bandwidth of all the receivers.

The cumulative distributions of end-to-end throughput for nodes from both before and after the transition in our 79-node experiment are presented in Fig. 12 and Fig. 14, respectively. Comparing with the case of a static multicast tree, it can be seen that the throughput of all the nodes are higher than in a tree. This could be baffling at first glance; however, it may very well be that a node does not utilize all its available bandwidth in a single tree, since a sub-optimal tree link may be chosen (which is likely due to the lack of knowledge of underlying network conditions and topology), or the network may become congested along the links in the tree. By dynamically monitoring and actively probing the network while disseminating data, a better performing topology may be progressively constructed. The experimental results confirm such conjectures that led to the design of *oEvolve*.

A further observation from our experimental results is the greater heterogeneity in the 79-node experiment compared to the 50-node experiment. By comparing Fig. 9 with Fig. 12, it is clear that with 79 nodes, there are *more* nodes achieving *higher* throughput than with 50 nodes. We believe that this may be because there are proportionately more high-bandwidth nodes that find each other to form high-bandwidth trees. These are encouraging results, since it indicates that with more nodes, *oEvolve* will generate topologies with better performance.

In summary, we have conducted a number of experiments over PlanetLab, a wide-area network testbed. Such experiments are not possible without our implementation of the message processing engine, and performance monitoring is not feasible without the implementation of the observer. Our algorithm, *oEvolve*, significantly improves the end-to-end throughput for *all* receivers, and are adaptive to varying network conditions caused by competing cross traffic. For those receivers with high available bandwidth, they are able to achieve proportionately high throughput, without adversely and unfairly affecting the throughput of lower bandwidth nodes.

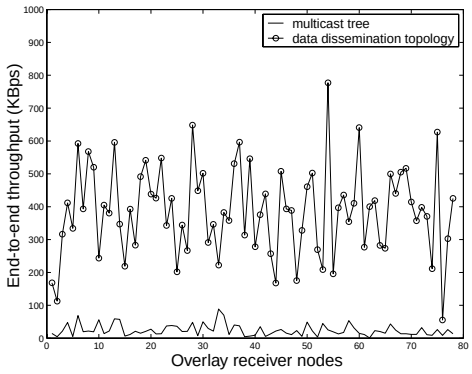


Fig. 11. Aggregate end-to-end throughput in the 79-node experiment in PlanetLab *before* the transitional event introducing cross traffic: (1) Multicast tree; (2) Data dissemination topology constructed by *oEvolve*.

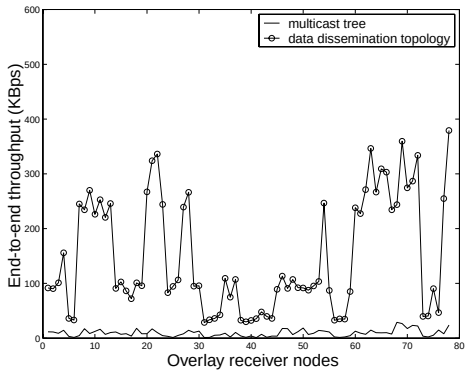


Fig. 13. Aggregate end-to-end throughput in the 79-node experiment in PlanetLab *after* the transitional event introducing cross traffic: (1) Multicast tree; (2) Data dissemination topology evolved by *oEvolve* after convergence.

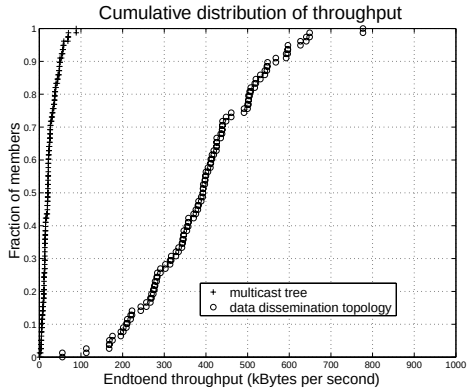


Fig. 12. Cumulative distribution of end-to-end throughput in the 79-node experiment in PlanetLab *before* the transitional event introducing cross traffic: (1) Multicast tree; (2) Data dissemination topology constructed by *oEvolve*.

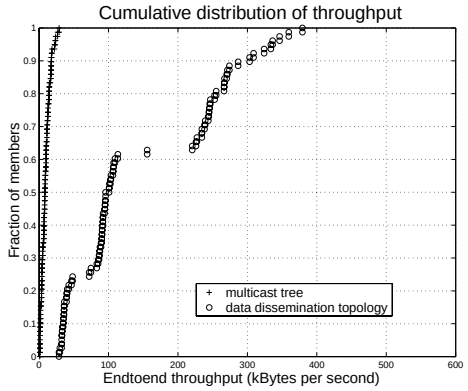


Fig. 14. Cumulative distribution of end-to-end throughput in the 79-node experiment in PlanetLab *after* the transitional event introducing cross traffic: (1) Multicast tree; (2) Data dissemination topology evolved by *oEvolve* after convergence.

VII. Related Work

Due to the difficulty of deployment of IP multicast, algorithms promoting application-layer overlay multicast have been proposed as remedial solutions, focusing on constructing a multicast tree. The common objective is to perform multicast with only unicasts between end hosts, and to maximize overlay performance. Narada [14], for example, first constructs an efficient mesh among members, and then construct a spanning tree of the mesh. More recently, there have been proposals of overlay multicast tree construction algorithms that aim to scale well to large group sizes, using tools including Delaunay Triangulations [15] and organizing members into hierarchies of clusters [16]. Algorithms have also been designed based on structured overlay networks (imposing data on specific nodes based on hash functions), examples include overlay multicast [17] based on CAN [18], as well as Scribe [19] based on Pastry [20]. These approaches may incur performance penalty, and may not be adaptive to dynamic network metrics, which is shown to be critical in overlay multicast routing, [21], [22].

oEvolve is more akin to two recent research papers that seek to utilize residual bandwidth availability by building multiple overlay multicast trees: CoopNet [23] and SplitStream [24]. CoopNet and SplitStream have proposed to utilize multiple trees to deliver striped data, using multiple description coding or source erasure codes. CoopNet proposes a centralized algorithm to facilitate using multiple multicast trees from different sources, and does not have explicit mechanisms to maximize bandwidth. In contrast, SplitStream has proposed a decentralized algorithm to construct a forest of multicast trees from a single source. The main idea is to build multiple *interior-node-disjoint* trees, which guarantee that each node serves as internal forwarding nodes in only one of the trees. SplitStream is developed based on Scribe, a tree-based multicast algorithm based on *structured* overlay networks.

oEvolve distinguishes from these previous work in many important aspects. *oEvolve*, as its name suggests, is designed from the ground up to *evolve* the data dissemination topologies beyond a single tree, by carefully evaluating the feasibility via available bandwidth measurements. The evolutionary nature also manifests itself when the network condition varies, as nodes join and withdraw from residual trees. In contrast, there are no provisions in both CoopNet and SplitStream regarding such evolutionary topologies based on performance measurements. In addition, *oEvolve* focuses on mechanisms and heuristics to address issues such as interference of flow constraints, measurements of residual bandwidth, and the detection and avoidance of deteriorating network conditions, none of which has been covered in any of the previous papers.

Compared to *oEvolve*, there also exist noteworthy disadvantages in both CoopNet and SplitStream. CoopNet is a centralized algorithm and thus introduces a central point of bottleneck and failures. SplitStream, on the other hand, is a distributed algorithm; but it is based on *structured overlay* multicast, which is generally not as generic as unstructured overlays that *oEvolve* assumes. In structured overlays, neighbor mappings are dictated by addresses from certain abstract coordinate spaces, which may incur performance penalties, and may not be versatile adapting to dynamic metrics such as available bandwidth.

Finally, Kostic *et al.* [25] and Byers *et al.* [26] have both proposed to construct an overlay mesh of concurrent data dissemination connections, each sending a (hopefully) disjoint set of data. As a node receives data from these connections and merges incoming data, throughput may be significantly improved due to the larger number of concurrent connections. Byers *et al.* has discussed the algorithmic details of merging differences from different downloading sources, while Kostic *et al.* has proposed an elaborate algorithm that allows nodes to send data to different points in the overlay, as well as to locate and recover missing data items. Both work had similar objectives to *oEvolve*, in the sense that they all seek to improve the bandwidth of data dissemination.

There are, however, significant differences comparing *oEvolve* to these approaches. First, while both [25] and [26] need to assume large or unlimited buffers at each overlay node in order to store elements of data to potentially serve others, *oEvolve* does not make this assumption. While this assumption is certainly valid when file-based rather than in-memory buffers are used, it unavoidably lacks the support for *delay-sensitive* data dissemination, such as real-time streaming of multimedia or stock quotes. Second, *oEvolve* shares the advantage of these approaches that the end-system available bandwidth is saturated, without the complexity of locating missing data items from a potentially large number of possible hosts — data may *only* arrive from upstream nodes in *oEvolve* residual trees. Finally, *oEvolve* enjoys its unique evolutionary nature with respect to the data dissemination topology. While [25] has briefly discussed heuristics such as pruning low-quality senders or receivers of data, such heuristics are optimizations in nature, and have not reached the maturity of step-by-step evolutionary mechanisms to achieve both fair and bandwidth-saturated data dissemination.

VIII. Concluding Remarks

In this paper, we have presented *oEvolve*, a distributed algorithm to significantly improve end-to-end bandwidth of data dissemination sessions. Due to the evolutionary nature of *oEvolve*, *oEvolve* is able to deliver a topology that is most suitable to the underlying network conditions, with respect to fairness, high end-to-end bandwidth, as well as fluctuating network conditions due to unknown physical topologies or cross traffic. With respect to the effectiveness and performance of our algorithm, we have undertaken both analytical studies and implementation on the *PlanetLab* wide-area overlay network testbed, and the evaluation results agree with our objective of high-bandwidth data dissemination in overlay networks.

References

[1] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A Blueprint for Introducing Disruptive Technology into the Internet," in *Proc. of HotNets-I*, 2002.

[2] R. Siamwalla, R. Sharma, and S. Keshav, "Discovering Internet Topology," in *Technical Report, Cornell University, http://www.cs.cornell.edu/skeshav/papers/discovery.pdf*, July 1998.

[3] News.com technology news, *http://news.com.com/2009-1033-269131.html?legacy=cnet*.

[4] IBM research, *http://www.research.ibm.com/wdm/motive/reason.html*.

[5] Agilent, *http://www.agilent.com/Feature/English/archive/C009.html*.

[6] D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1987.

[7] Z. Wang and J. Crowcroft, "Quality of Service Routing for Supporting Multimedia Applications," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, September 1996.

[8] M. Jain and C. Dovrolis, "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput," in *Proc. of ACM SIGCOMM*, 2002.

[9] J.J. Garcia-Luna-Aceves, "A Distributed Loop-Free, Shortest-Path Routing Algorithm," in *Proc. of IEEE INFOCOM*, 1988.

[10] M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1986.

[11] N. Garg, R. Khandekar, K. Kunal, and V. Pandit, "Bandwidth Maximization in Multicasting," in *Proc. of the 11th Annual European Symposium on Algorithms*, September 2003.

[12] S. McCanne and V. Jacobson, "Receiver-driven Layered Multicast," in *Proc. of ACM SIGCOMM*, 1996.

[13] "iOverlay: A Generic and High-Performance Substrate for Overlay Algorithm Implementations," in *Technical Report, University of Toronto*, October 2003.

[14] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communications*, pp. 1456–1471, October 2002.

[15] J. Liebeherr, M. Nahas, and W. Si, "Application-Layer Multicasting With Delaunay Triangulation Overlays," *IEEE Journal on Selected Areas in Communications*, pp. 1472–1488, 2002.

[16] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. of ACM SIGCOMM*, August 2002.

[17] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "Application-level Multicast Using Content-addressable Networks," in *Proc. 3rd Int. Workshop on Networked Group Communication (NGC '01)*, London, UK, 2001.

[18] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," in *Proc. of ACM SIGCOMM*, August 2001, pp. 149–160.

[19] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, "Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure," *IEEE Journal on Selected Areas in Communications*, pp. 1489–1499, October 2002.

[20] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems," in *Proc. IFIP/ACM Middleware 2001*, November 2001.

[21] S. Shi and S. Turner, "Multicast Routing and Bandwidth Dimensioning in Overlay Networks," *IEEE Journal on Selected Areas in Communications*, pp. 1444–1455, October 2002.

[22] M. Faloutsos, A. Banerjea, and R. Pankaj, "QoSMIC: Quality of Service sensitive Multicast Internet protoCol," in *Proc. of ACM SIGCOMM*, August 1998.

[23] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Distributing Streaming Media Content Using Cooperative Networking," in *Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2002)*, Miami Beach, Florida, May 2002.

[24] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments," in *Proc. of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, October 2003.

[25] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," in *Proc. of ACM SOSP*, 2003.

[26] J. Byers and J. Considine, "Informed Content Delivery Across Adaptive Overlay Networks," in *Proc. of ACM SIGCOMM*, August 2002.