Efficient Large Scale Content Distribution

Danny Bickson, Dahlia Malkhi and David Rabinowitz*

Abstract

This paper considers the problem of supporting high-bandwidth, scalable application level multicast. It provides a formal treatment of the problem which takes into consideration network distances. It presents the Julia locality-aware content dissemination scheme, a highly efficient and fast method whose distribution is both time-aware and proximity-aware. A formal comparison of Julia with several reference protocols is given, showing that Julia is significantly more efficient than previous works.

keywords: locality aware networks, large scale content distribution, high bandwidth data dissemination, application multicast.

1 Introduction

We consider the problem of supporting high-bandwidth, scalable application-level multicast. This problem is well motivated by the increasing popularity of Internet-wide information sharing facilities, e.g., content distribution networks such as KaZaA, of multimedia multicast services, and chat groups of various sorts.

Ideally, multicast would be supported at the network level. This would allow optimal routing and avoid duplicity, two measures of multicast efficiency that were identified in [7] (and called there *relative delay penalty* and *physical link stress*). However, there are currently no existing, deployed protocols that offer a full solution. For example, while IP Multicast provides standard infrastructure for one-to-many dissemination, it offers no reliability. SRM [9] enhances IP-multicast with controlled and randomized retransmissions. Still, SRM has scalability problems, and in addition, leaves open the issue of maintaining meta information, such as membership views for IP-multicast groups.

This has led to a vast amount of work that tackles application-level multicast by building overlay networks over which messages are multicast. Several works are based on a tree topology for data dissemination from a source to its destinations. The main advantage of tree-based multicast is an exponential dissemination: While the root is busy sending copies of a file to its children, the first ones to receive the content start pushing it to their descendents, simultaneously with the root. Recursively, the content is being pushed down the tree in parallel. So a tree solution fully distributes content to all participants in logarithmic time. Multicast systems based on trees include the Bayeux network [19] which is built over Tapestry [18], and others like [7]. These schemes vary in their choice of links, and the maintenance of their data. However, a common problem in all of these schemes is that inner nodes in a tree have higher importance, and consequently, failures in inner nodes may cut a large fraction of the system from its input. In addition, inner nodes suffer higher communication load than the leaves.

More recent solutions address both of these problems via striping. The content is split into pieces (quite possibly with some redundancy through error correcting codes). The source pushes the pieces of the file to an initial group of nodes, each of which becomes a source of a distribution tree for its piece, and pushes it to all other nodes. SplitStream [5] employs the Pastry routing overlay in order to construct multiple trees,

*School of Computer Science and Engineering, The Hebrew University of Jerusalem. Email: {daniel51,dalia,dar}@cs.huji.ac.il

such that each participating node is an inner node in only one tree. It then supports parallel download of stripes within all trees. Bullet [11] takes a less structured approach, and one which attempts to maximize the download bandwidth available for each recipient. It achieves this by letting each node select on the fly nodes from which to download stripes. The selection by each node is done from among a uniformly drawn partial view of the system, as provided by an underlying view-gossip protocol called RanSub [10]. FastReplica [6] creates a full graph between the participating servers. These works demonstrate clearly the advantages of data striping, i.e., of simultaneously exchanging stripes of data, over a tree-based dissemination of the full content.

Taking this successful approach a step further, the first contribution of our work is a clean and formal problem statement, that allows us to assess the efficiency of different solutions in sterile settings.

The common problem model for all of the above schemes is as follows. There are n nodes that contact a single source and request a file Foo from it. The file Foo has $|Foo|$ bits. During this initial contact, the source may inform the participants about the identities of all other participants. The participants then engage in a download protocol.

The network provides full connectivity between nodes. We assume that in one time *unit*, each node can transmit one bit to one other node; and a node may receive (and process) any bits sent to it. We do take into account the *distance* that bits travel. To this end, we assume that there is an associated distance metric between pairs of nodes.

This problem model is naturally somewhat over-simplistic, in several respects. First, it assumes that all participants arrive simultaneously to ask for the file. In practice, any algorithm developed for this model will be applied *gradually*, i.e., each node will arrive at a different time and will learn from the source about existing participants. Nevertheless, we emphasize that having a clean problem statement allows us to formally reason about solutions, and have a good basis from which deployment is derived. It should also be stressed that the source is **not** a priori determined, i.e., we strive for solutions that provide all-to-all multicast, but the analysis focuses on a single file's dissemination.

The timing model ignores asynchrony, scheduling delays, and variability in bandwidth. This is done for simplicity, and it should be clear that all of the algorithms we analyze work correctly in asynchronous settings. The timing model is used purely for analysis purposes, in order to compare different solutions. Distances (or latencies) are taken into account when considering edge-weights, but not in the time unit. This simplification does not impair the analysis significantly, since large messages incur a single additive latency over their entire content.

Under this synthetic problem statement, we define the following measures of efficiency for the content distribution problem.

1. **Time:** The total time until all of the nodes receive the file Foo.
2. **Work:** The sum over all messages employed in the protocol of <message-size> $\times$ <distance>.[1]
3. **Connectivity:** The number of connections per-node needed for performing the download, maximized over all n nodes (except for the source).
4. **Fair-Sharing:** The ratio between the number of bits transmitted by a node to the number of bits received by it, maximized over all n nodes (except for the source).

We analyze several known algorithms according to these measures, including a naive multicast tree, SplitStream [5] and FastReplica [6]. To the best of our knowledge, ours is the first formal treatment of the

[1] A similar approach was taken in [7] where the *Resource Usage* is defined to be the sum of latencies multiplied by the link stress. (Link stress is the number of identical messages which travel the same link).

content distribution problem using precise measures that takes into account network distances, as well as timing. We view having a precise and clear problem statement as an important step in the design of real systems. Without it, it is hard to compare different approaches, and it is difficult to extrapolate from one experimental settings to another.

Moreover, our analysis reveals surprising results. In particular, we show that tree-based dissemination, taking logarithmic duration, this is **not** optimal: Constant Time suffices for a full n-way content distribution.

Regarding Work, the situation is even worse. We demonstrate realistic network geometries under which all of the above algorithms suffer Work load that is almost linear in n. In general, the optimum is topology-dependent, but for the networks we consider, it is logarithmic. This is quite far from the Work achieved by these schemes.

Our second contribution is a novel protocol, Julia, that improves on previous work in both Time and Work. Julia uses file striping, and exchanges file pieces in a Time-aware and Work-aware manner. The idea of Julia is a 'divide-and-conquer' method. We view the network as being built hierarchically. Intuitively, think of the network as divided into 'continents', each one comprising of 'countries', then 'districts', and so on. Initially, we arrange to have a single copy of each file-piece delivered into each one of the continents (though no single node in the continent need have all pieces). In the next stage, we arrange an exchange independently within each continent, so that each country obtains copies of all file pieces. And so on in a recursive manner. In the end, each participant obtains all file pieces, i.e., the file. The Time of our algorithm is optimal, $2|Foo|$. The Work is logarithmic, which is optimal for certain realistic network models. Both of these are substantial improvements over previous methods. All of this is done while preserving fair-sharing and using low (logarithmic) connectivity.

Our formulation of the problem and its solution are only the first step. In reality, adaptations to the formal algorithm need to be made. This stems from the fact that the overlay network structure needs to be deployed incrementally, without central coordination. Therefore, the information held by each node is neither complete, nor accurate. Furthermore, protocols must have sufficient flexibility to cope with high churn.

Our deployment approach is highly flexible and fault tolerant. More concretely, whereas our protocol requires a node to select links of certain exact distances and identities, in practice we plan for each node to have only rough information about a few nodes in its vicinity; about a few nodes in the medium distance; and about very far nodes. Among these, the node chooses its partners for exchanging data in the content delivery protocol. Work is underway to evaluate our protocol in realistic wide-area network scenarios, using the Planet-lab infrastructure [8] and the Evergrow project's test-bed [2].

2 Problem Statement

We begin with a precise statement of our network assumptions, and the goals we aim at.

The network contains n nodes, one of them being the *source* node. Nodes communicate via a directed, weighted full graph. The weight of an edge indicates the distance between two nodes. The schemes studied in the paper operate correctly over any set of weights. For the purpose of analysis, we study a simple directed ring. The ring diameter is D, and it has n uniform grid points at positions $D/n, 2D/n, \ldots, (n-1)D/n$, D. The distance between two points is their clockwise distance, i.e., each node has neighbors at distances $D/n, 2D/n, \ldots, (n-1)D/n, D$.

Time is measured using a logical *time unit*. During one unit, every node sends one bit to one other node it selects, and every node receives and processes all the bits sent to it. This model ignores the distance (or edge weight) over which the bit is sent. The reason we choose to model time this way is that in real internets, the bandwidths are more or less uniform (though latencies are not). Since we envision sending

large amounts of data, the initial latencies of messages would be negligible in the computation, and so we treat all bits as if they flow at the same rate.

The time measurement starts simultaneously at all nodes. At the time the protocol starts, every node recognizes all other $n-1$ nodes and knows the edge-distances to them. The nodes contact a single source and request a file Foo from it. The file Foo has $|Foo|$ bits.

The simultaneous start, and the global knowledge assumption, are of course both unrealistic. They are used in order to form intuition about the behavior of different protocols and are utilized for analysis purposes only. In practice, all of the algorithms may work asynchronously, and utilize partial knowledge of the network. Part of our on-going work on Julia's deployment addresses these issues, and contains mechanisms for propagation of configuration information, routing tables maintenance, and distance assessment.

Measures: Many previous application-level multicast protocols did not consider distance but only the sequential time it takes to disseminate content among a set of nodes. Thus, they may unnecessarily load the network by sending the same content over long-haul links. As a simple example, consider a multicast-tree. It may appear to distribute content is optimal time (though poor load balancing and fault tolerance, but these are besides the point). However, it may perform poorly in real network settings because the same content travels from one continent to another and back again due to its ignorance of distances.

Clearly, coming out of a source and reaching n destinations, each bit must travel n times. But the question is, to what distance does each bit travel? We define a measure of multicast locality, *Work*. The Work measure is defined as the total amount of bits sent over the network, each multiplied by the edge-weight it transfers over.

Coupled with Work are several additional measures of efficiency. *Time* measures the total distribution time-units from beginning and until the last node receives the content in full.

For practical reasons, it is desirable to have each node maintain connections to a reasonable number of other nodes. This is captured by the *Connectivity* measure, defined as the maximum number of connections any node has during the protocol, except the source.

Finally, we define a measure of load balance called *Fair-Share*. This measures the ratio between how much data a node serves and how much it obtains from the protocol, maximized over all nodes except for the source.

3 Reference Algorithms

In this section, we briefly describe the content distribution algorithms we selected for analysis.

The first algorithm is a naive application-level multicast tree of degree k. In this scheme, the source node which has the complete file to distribute is the tree root. Each non-leaf node has k children. Each node transfers the complete file to all its children one by one. The depth of an n-node tree of degree k is $d = \log_k(n)$.

Bayuex [19] relies on Tapestry [18] for location and routing services. In order to publish a file, the source advertises using flooding a tuple which contains the semantic name of a multicast session and a unique id. This tuple is hashed to obtain a node identifier which becomes the session root node. Each node can join this multicast session by sending a message to the root. Nodes along the way maintain membership information, so that a multicast tree is formed in the reverse direction. The file content (and any updates) are flooded down the tree. Bayuex leverages the locality properties of Tapestry to sending the file from the root to members with per-member cost that is proportional to the actual network distance. Nevertheless, as we shall see below, the simultaneous dissemination to multiple members is not necessarily Work optimal.

SplitStream [5], built over the Pastry overlay [15] strives to obtain load balancing between multicast nodes. It achieves that by splitting the published content to several parts, called stripes, and publishing each

part separately. Each stripe is published using a tree-based multicast. The workload is divided between the participating nodes by sending each stripe using a different multicast tree. Load balance is achieved by carefully choosing the multicast trees so that each node serves as an interior node in at most one tree. This reduces the number of "free riders" who only receive data. Like Bayuex, SplitStream enjoys the locality of the underlying Pastry overlay, so that each individual route between a pair of nodes is proportional to their network distance. This again does not yield Work optimality.

FastReplica [6] is based on a full graph topology. The algorithm consists of two phases. In the first phase, the source node partitions the file into k distinct pieces, and sends each node one piece. In the second phase, the nodes exchange their pieces using an all-to-all exchange. This construction can be nested recursively, by building a tree of several levels of full graphs. When the first level completes those two stages, each of the nodes becomes a root and distributes the file recursively to k full graphs containing k^2 nodes and so on.

Other works which have some locality considerations are Narada [7], Zigzag [16] and TMesh [17] where a spanning tree rooted at the source heuristically approximates the underlying network structure. However, the created topology does not necessarily achieves the optimal.

4 The Julia Protocol

Our protocol is designed to simultaneously achieve several challenging goals. First, it strives to minimize download Time. The second goal is to minimize Work. This should be done by exploiting locality and distributing content along short communication paths.

In addition to minimizing Time and Work, an underlying goal of our design is to balance the load between participating nodes, i.e., have Fare-Sharing as close to 1 as possible. This is done so as to prevent free riding, which was identified in [4] is a key problem of unbalanced content exchange networks. They show that almost 70% of the peers in the Gnutella network are users who share no files.

Just as in SplitStream [5] and Bullet [11], the basic idea of our protocol is to divide the file Foo between the set of n participants, and then utilize a mesh of links in order to simultaneously exchange pieces of Foo among them. Unlike previous protocols, our design emphasizes locality awareness during the exchange phase, in order to minimize Work.

The exchange topology we require can be constructed over several existing locality-aware overlays, including the scheme by Plaxton et al. [13], Pastry [15], Tapestry [18] and LAND [3]. More concretely, we require the following property. Let u be some node. Node u should have a *level-1 link* to a node that is farther away from u than at least half of the nodes. For the protocol to be balanced, no other node should have the same node as its level-1 target. Likewise, we need a unique *level-2 link* to distance larger than a quarter of the network. And so on, until we reach a link to the closest node. Thus, there are $\log(n)$ links.

In the locality aware networks cited above [13, 15, 18, 3], this property is roughly maintained using node identifiers. This works as follows. Node identifiers are selected at random, hence, by assumption, identifiers are uniformly dispersed in the network. The level-1 link of a node with a certain identifier x is the link of x that matches x in all bits except the least significant bit (LSB). There is only one such target node in the network, and by uniformity, this node is expected to be found farther away from x than at least half of the nodes. The level-2 link of x is the closest node that matches x in all but the last two LSBs. Since there are four such nodes in the network, and they are scattered uniformly, the distance of this link is expected to be greater than one fourth of the network. And so on. The level-$(\log(n))$ link goes to the closest node.

It is worth noting that although we could build our protocols over the existing infrastructure of the above systems, our implementation of locality-aware links is planned to be significantly less structured, and much more flexible. Our links will be chosen based on distance estimation, which is the subject of intensive current experimental work. Within each distance-class, we randomly select a target, and maintain

information about several potential replacements. As messages keep flowing in the network, nodes may learn of more suitable target nodes and replace certain links, or may learn of link failures and reassign them. Due to the approximate nature of the deployment, several nodes might choose the same target node for a particular link. Nevertheless, we expect not to have high duplicity and will deal with it in an ad hoc manner. In this way, there is sufficient flexibility and redundancy so that maintaining the network in scalable settings is manageable, and coping with churn is simple.

A useful intuition on the topology of Julia is that of a locality-aware hypercube, as depicted in Figure 1. The two main faces of the hypercube (front and back) are the most geographically distant from one another. By way of an example, we may have one face contain all the nodes in the north American continent, and the other face has European nodes. Level-1 links of the nodes on either face 'cross the atlantic' to their half-network neighbors away.

Within each face, nodes are split geographically along the vertical split (so left nodes are on the west coast, and right nodes on the east coast). Level-2 links cross between the left and the right sides of each face. And so on.

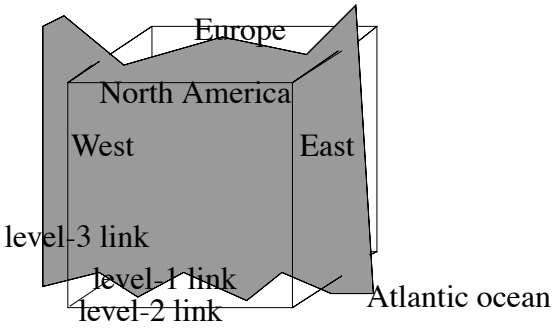

Figure 1: Julia's locality-aware topology.

The content dissemination protocol works over the exchange overlay as follows. Recall that as a first step, the source of Foo sends pieces of Foo to all participants. Each piece has size $\frac{|Foo|}{n}$. [2]. The remainder of the protocol consists of $\log(n)$ logical rounds. In round i, each node exchanges all the pieces is holds with its level-i link partner. In this way, round 1 consists of exchanges of single pieces, each of size $|Foo|/n$, sent over half the network away. And generally, round i consists of exchanges of 2^i pieces, totaling in size $|Foo|2^i/n$, over a distance of $1/2^i$. In the last round, nodes exchange with their closest neighbors data of size $|Foo|/2$. Figure 2 depicts the $\log(n)$ exchange-rounds of our protocol.

4.1 Protocol Properties

In each round of the protocol, every pair of nodes that exchange data holds disjoint pieces of Foo. Thus, the pair of nodes exchange **all** the information they accumulated up to this round. As a result, the protocol enjoys both simplicity and complete symmetry. In particular, all nodes experience the same load, and no node can be a free rider.

Another important feature is locality awareness. This is manifested in the fact that the size of an exchange is reversely related to its distance. In fact, along the longest edges, going half the network away, only single pieces are sent; edges going a distance fourth of the network suffer a transfer of a pair of pieces each; and so on. The largest data exchange, of size $|Foo|/2$, is carried between close neighbors. In this way, most of the download is done locally.

[2] A straight-forward extension of our scheme is to employ an erasure code such as IDA [14] to transform Foo into n parts, such that $n - t$ parts, for some parameter t, suffice to reconstruct Foo.

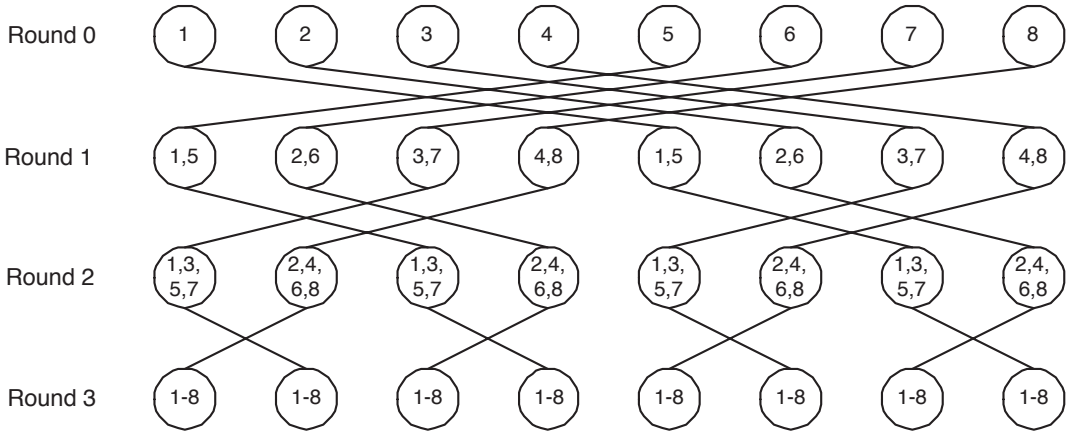

Figure 2: *Rounds* $1..\log(n)$ *of the content distribution protocol,* $n = 8$. *The numbers inside each node denote the file pieces it holds at the beginning of the round.*

Finally, all the nodes are busy downloading all the time, and no nodes are idle waiting for internal bottlenecks to complete. Thus, the time for the download of the full file is optimized, and is equal among all nodes.

5 Performance Analysis of the Protocols

In this section, we provide a formal analysis of all of the above content delivery protocols according to our performance measures, Work, Time Connectivity and Fair-Sharing. We begin with the Julia protocol, and continue with the reference algorithms. In all calculations, we assume that n is a power of 2, or else we will need to mark the tree depth to be $\lceil \log(n) \rceil$ instead of $\log(n)$.

For time, the optimal lower bound presented in [1] is $|Foo|1 + log_k(n)$. The intuition for it is very simple. In order to get the file out of the source, we need a time of $|Foo|$. For the last bit which got out of the source, using recursive doubling we need another $|Foo| \log(n)$ rounds.

For Time, in Julia the total download time is calculated as follows. The first $|Foo|$ time units are needed for transferring the file from the source to all nodes, each node getting one piece of the file. Then, in logical round i, all pairs simultaneously exchange $2^{(i-1)}$ pieces of size $\frac{|Foo|}{n}$ each. The time (including initialization) is: $|Foo| + \sum_{i=1..\log(n)} \frac{2^{(i-1)}|Foo|}{n} = 2|Foo|$. A straight forward extension is to begin pipelining the the information before the source finished distributing all the data. That can be done in time which is very near optimal as shown in [1]. However, we do not consider the time only as one of the parameters we want to optimize. We want to minimize the Work as possible as well.

For Connectivity, the number of edges in our exchange overlay network is $n \log(n)$.

The Fair-Sharing ratio of Julia is precisely 1 by design.

The Work in our protocol is calculated as follows: At the initial phase, each node receives from the source one piece of size $\frac{|Foo|}{n}$ over links of average length $\frac{D}{2}$. Thus over n nodes, we get a total of $\frac{|Foo|D}{2}$. Regarding the algorithm, we sum up $\log(n)$ protocol rounds. In round i, exchanges are carried over a distance of $\frac{D}{2^i}$. Each data exchange between a pair of nodes carries $2^{(i-1)}$ pieces of size $\frac{|Foo|}{n}$ each. The total exchanged data in the network in round i is $n\frac{2^{(i-1)}|Foo|}{n}$. The resulting Work is:

$$|Foo|\frac{D}{2} + \sum_{i=0}^{\log(n)} n\frac{D}{2^i}\frac{2^{(i-1)}|Foo|}{n} = |Foo|\frac{D}{2}(\log(n)+1)\ .$$

Application multicast tree. An application multicast tree is an n-node tree of degree k, whose depth is $d = \log_k(n)$. Multicast is done by forwarding Foo in its entirety to inner tree nodes, each of them pushing Foo to its children one after the other.

The Connectivity of a tree is $n-1$. The Fair-Sharing ratio is k, since inner tree nodes have in-degree 1 and out-degree k.

For Time, the worst download completion time is $|Foo|kd$ since the last leaf node has to wait until all of its parents obtain the complete file. The Work in the tree is divided into two cases. If the tree is randomly constructed, then each edge has an average length of $\frac{D}{2}$, and the total Work is $|Foo|(n-1)\frac{D}{2}$. However, it is more interesting to analyze a construction of tree over one of the locality aware multicast networks like Scribe [12]. Since the number of tree edges in each level is exponential in k, the best Work is achieved when we select the edges at level i of the tree to have weight $D/2^i$. Thus, the leaf edges are the shortest edges, of distance $\frac{D}{2^{log_k(n)}}$. The Work in this case is calculated as follows: For each level i of the tree, for $i = 1..d$, we have $k^{(i-1)}$ tree nodes, and k^i outgoing edges, each of length $\frac{D}{2^i}$. Each node sends $|Foo|$ bits of data to its k children. The Work is given by the following sum:

$$\begin{aligned}
\sum_{i=1}^{d} k^i(\frac{D}{2^i})|Foo| &= |Foo|D\sum_{i=1}^{d}(\frac{k}{2})^i && (1)\\
&= |Foo|D\frac{k}{2}\frac{\left(\frac{k}{2}\right)^{(\log_k(n))}-1}{\frac{k}{2}-1} && (2)\\
&= |Foo|D\frac{k(n^{\left(1-\frac{1}{\log_k(n)}\right)}-2}{(k-2)} && (3)\\
&\geq \frac{|Foo|Dn^{(1-\frac{1}{d})}}{2} && (4)
\end{aligned}$$

SplitStream. The SplitStream construction consists of k multicast trees, each of which disseminates a piece (called a *slice* in SplitStream) of size $\frac{|Foo|}{k}$ to all n nodes. The tree depth is $d = \log_k(n)$. The worst download time is constructed of $|Foo|$ time units needed to transfer the file from the source into the k subtrees and another $k\frac{|Foo|}{k}kd = |Foo|\log_k(n)$ needed for the actual protocol. (We assume the transfer is done using pipelining where the first trees already start to forward the data even before the last trees got it). The total is $k|Foo|(\log_k(n)+1)$ which is the optimal lower bound.

For Connectivity, the number of edges is $(n-1)k$ edges since each node has one parent node at each tree, and there are k such trees. (The choice of k recommended in [5] is 16.) The worst Fair-Sharing ratio is 1 since the k trees completely balance the load.

The Work in the SplitStream protocol is calculated as follows: For initialization of the k tree roots with distinct pieces, we have Work of transferring parts of size $\frac{|Foo|}{k}$ across links of average length of $\frac{D}{2}$. We have k such tress, so the total is $\frac{|Foo|}{k}k\frac{D}{2} = |Foo|\frac{D}{2}$. Next, we have k trees, each composed of d levels. In each tree, level i, for $i = 1..d$, has $k^{(i-1)}$ nodes and k^i edges, each of distance $\frac{D}{2^i}$.

$$k\sum_{i=1}^{d}\frac{D}{2^i}k^i\frac{|Foo|}{k} = |Foo|D\sum_{i=1}^{d}\left(\frac{k}{2}\right)^i \quad (5)$$

$$= |Foo|D\frac{k}{2}\frac{\left(\frac{k}{2}\right)^{(\log_k(n))}-1}{\frac{k}{2}-1} \quad (6)$$

$$= |Foo|D\frac{k(n^{\left(1-\frac{1}{\log_k(n)}\right)}-2}{k-2} \quad (7)$$

$$\geq \frac{|Foo|Dkn^{(1-\frac{1}{d})}}{2} \quad (8)$$

Adding the initialization phase Work we get:

$$\frac{|Foo|D(kn^{(1-\frac{1}{d})}+1)}{2}$$

FastReplica. In the FastReplica protocol (using the suggested recursive construction), each parent has k children. There are $d = \log_k(n)$ levels. Each set of siblings is connected among itself in a full graph topology. The worst download time is calculated as follows: $|Foo|\log_k(n)$ is the initial distribution phase time. The replication phase takes another $\log_k(n)(k-1)\frac{|Foo|}{k}$, so in total the Time is $(2-\frac{1}{k})|Foo|\log_k(n)$.

The number of edges required is $(n-1)+(n-1)\frac{k-1}{2} = (n-1)\frac{k+1}{2}$. The worst Fair-Sharing ratio is 1 since in the distribution phase the source transfers the file only once, and in the replication phase each nodes sends the same amount of data it receives. The total Work is calculated by the two protocol phases: In the distribution phase, the source node sends a part of size $\frac{|Foo|}{k}$ to all its children. At level i there are k^{i-1} such source nodes, each one of them transfers a part of size $\frac{|Foo|}{k}$ to its k children. We assume that as in the locality-aware multicast tree, the edges at level i are of distance $D/2^i$. We also assume that the all-to-all links at level i have distance $\frac{D}{2^{(i+1)}}$. Work is calculated as follows:

$$\sum_{i=1}^{d}k^i\frac{D}{2^i}\frac{|Foo|}{k} = \frac{|Foo|D}{k}\sum_{i=1}^{d}(\frac{k}{2})^i\,.$$

Thus, we have exactly $\frac{1}{k}$'th of the Work in the SplitStream protocol. In the replication phase, the children exchange the information they get in the distribution phase among themselves, using the all-to-all links. The Work is:

$$\sum_{i=1}^{d}(k-1)k^i\frac{D}{2^{(i+1)}}\frac{|Foo|}{k} = \frac{|Foo|D(k-1)}{2k}\sum_{i=1}^{d}(\frac{k}{2})^i\,,$$

and that is $\frac{k-1}{2k}$ times the SplitStream work. The total Work is bounded below by

$$\left(\frac{1}{k}+\frac{k-1}{2k}\right)\frac{|Foo|Dkn^{(1-\frac{1}{d})}}{2} \geq \frac{|Foo|Dkn^{(1-\frac{1}{d})}}{4}\,.$$

The results are presented in Table 1 below.

As evident from the table, our protocol achieves the best download time, independent of n. All other protocols have a logarithmic download time. Our protocol is also the most economic in terms of the total

	Time	Connectivity	Work	Fair-Sharing
Application multicast tree	$\|Foo\|k\log_k(n)$	$n-1$	$\|Foo\|\frac{D}{2}(n-1)$	k
Multicast tree over Scribe	$\|Foo\|k\log_k(n)$	$n-1$	$\Omega(\|Foo\|Dn^{1-\alpha})$	k
SplitStream Protocol	$\|Foo\|k(\log_k(n)+1)$	$(n-1)k$	$\Omega(\|Foo\|Dn^{1-\alpha})$	1
FastReplica Protocol	$(2-\frac{1}{k})\|Foo\|\log_k(n)$	$(n-1)\frac{k+1}{2}$	$\Omega(\|Foo\|Dn^{1-\alpha})$	1
Our protocol	$2\|Foo\|$	$n\log(n)$	$\Theta(\|Foo\|D\log(n))$	1

Table 1: *Comparison of various protocols.* n *is the total number of nodes,* D *is the network diameter,* $|Foo|$ *is the downloaded file size. In the multicast tree,* k *is the outgoing degree of the interior nodes, in the SplitStream protocol,* k *is the number of slices and in FastReplica* k *is the mesh size. In all cases* $\alpha = \frac{1}{\log_2(k)}$.

Work. All of this is achieved while preserving Fair-Sharing 1 of SplitStream and FastReplica. We pay in a logarithmic factor in the Connectivity relative to the other protocols.

We believe that in practice, the logarithmic increase in connectivity is quite manageable. In fact, Connectivity $n\log(n)$ is comparable to kn for reasonable sizes of n, e.g., when $k = 16$.

6 Future Work

We are currently in the process of building a prototype implementation of our proposed protocol. Our current implementation runs over the Planetlab test-bed for worldwide experiments [8]. In the near future, we plan to also use for our tests the Evergrow infrastructure [2], that will contain several clusters of computers residing in different geographic locations. Our deployment will tackle several challenging practical issues.

First, our simplistic analysis has assumed a uniform node dispersal. However, in the real world nodes are not perfectly distributed. Our experiments should provide insight as to how content distribution protocols perform in real settings, and feed back guidance as to how to improve their design.

Second, nodes in the network do not know the actual network distances. Our ongoing effort aims to approximately learn the network metrics in a dynamic and adaptable manner. We currently measure several distance measures, composed of the following parameters: Round-trip message transmission time, number of router hops to destination, link incoming and outgoing bandwidth, , IP address bit difference and DNS longest common suffix.

We will use these approximate distance data to form several layers of clustering that will be used in forming our multi-level links. Our main goal is to prove that using locality improves the practical download time of large content distributed to a large number of nodes. We further want to show that the actual work done by the participating nodes is well balanced.

Finally, we will test the behavior of our algorithm in face of faults and dynamic joins and departures, and fine tune it accordingly.

References

[1]

[2] Evergrow eu project.

[3] I. Abraham, D. Malkhi, and O. Dobzinski. LAND: Stretch $(1 + \varepsilon)$ locality aware networks for DHTs. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA04)*, 2004.

[4] E. Adar and B. Huberman. Free riding on gnutella. *First Monday*, 5(10), October 2000.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in a cooperative environment. In *SOSP'03*, October 2003.

[6] L. Cherkasova and J. Lee. Fastreplica: Efficent large scale distribution within content delivary networks. 2003.

[7] Y. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *In Proceedings of ACM SIGMETRICS, Santa Clara, CA, pp 1-12*, June 2000.

[8] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. Planetlab: an overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[9] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. In *IEEE/ACM Transactions on Networking, December 1997, Volume 5, Number 6, pp. 784-803*.

[10] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat. Using random subsets to build scalable network services, 2003.

[11] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *ACM SOSP*, October 2003.

[12] A.-M. K. M. Castro, P. Druschel and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications)*, 2002.

[13] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the Ninth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA 97)*, pages 311–320, 1997.

[14] M. O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

[15] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.

[16] D. A. Tran, K. A. Hua, and T. T. Do. A peer-to-peer architecture for media streaming. 2003.

[17] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *In Proceedings of Fourth International Workshop on Networked Group Communication (NGC)*, Obtober 2002.

[18] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.

[19] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, 2001.