FastReplica: Efficient Large File Distribution within Content Delivery Networks

Ludmila Cherkasova
Hewlett-Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94303
cherkasova@hpl.hp.com

Jangwon Lee
University of Texas at Austin
Austin, Texas 78712
jangwlee@ece.utexas.edu

Abstract. *In this work, we consider a large-scale distributed network of servers and a problem of content distribution across it. We propose a novel algorithm, called FastReplica, for an efficient and reliable replication of large files in the Internet environment. There are a few basic ideas exploited in FastReplica. In order to replicate a large file among n nodes (n is in the range of 10-30 nodes), the original file is partitioned into n subfiles of equal size and each subfile is transferred to a different node in the group. After that, each node propagates its subfile to the remaining nodes in the group. Thus instead of the typical replication of an entire file to n nodes by using n Internet paths, connecting the original node to the replication group, FastReplica exploits $n \times n$ Internet paths within the replication group where each path is used for transferring $\frac{1}{n}$-th of the file. We design a scalable and reliable FastReplica algorithm which can be used for replication of large files to a large group of nodes. The new method is simple and inexpensive. It does not require any changes or modifications to the existing Internet infrastructure, and at the same time, it significantly reduces the file replication time as we demonstrate through experiments on a prototype implementation of FastReplica in a wide-area testbed.*

1 Introduction

Content Delivery Networks (CDNs) are based on a large-scale distributed network of servers located closer to the edges of the Internet for efficient delivery of digital content including various forms of multimedia content. The main goal of the CDN's architecture is to minimize the network impact in the critical path of content delivery as well as to overcome a server overload problem that is a serious threat for busy sites serving popular content.

For typical web documents (e.g. html pages and images) served via CDN, there is no need for active replication of the original content at the edge servers. The CDN's edge servers are the caching servers, and if the requested content is not yet in the cache, this document is retrieved from the original server, using the so-called *pull model*. The performance penalty associated with initial document retrieval from the original server, such as higher latency observed by the client and the additional load experienced by the original server, is not significant for small to medium size web documents.

For large documents, software download packages and media files, a different operational mode is preferred: it is desirable to replicate these files at edge servers in advance, using the so-called *push model*. For large files it is a challenging, resource-intensive problem, e.g. media files can require significant bandwidth and download time due to their large sizes: 20 min media file encoded at 1 Mbit/s results in a file of 150 MBytes.

The sites supported for efficiency reasons by multiple mirror servers, face a similar problem: the original content needs to be replicated across the multiple, geographically distributed, mirror servers.

While transferring a large file with individual point-to-point connections from an original server can be a viable solution in the case of limited number of mirror servers (tenths of servers), this method does not scale when the content needs to be replicated across a CDN with thousands of geographically distributed machines.

Currently, the three most popular methods used for content distribution (replication) in the Internet environment are:

- *satellite distribution*,
- *multicast distribution*,
- *application-level multicast distribution*.

With *satellite distribution* [8, 21], the content distribution server (or the original site) has a transmitting antenna. The servers, to which the content should be replicated, (or the corresponding Internet Data Centers, where the servers are located) have a satellite receiving dish. The original content distribution server broadcasts a file via a satellite channel. Among the shortcomings of the satellite distribution method are that it requires special hardware deployment and the supporting infrastructure (or service) is quite expensive.

With *multicast distribution*, an application can send one copy of each packet and address it to the group of nodes (IP addresses) that want to receive it. This technique reduces network traffic by simultaneously delivering a single stream of information to hundreds/thousands of interested recipients. Multicast can

be implemented at both the data-link layer and the network layer. Applications that take advantage of multicast technologies include video conferencing, corporate communications, distance learning, and distribution of software, stock quotes, and news. Among the shortcomings of the multicast distribution method are that it requires a multicast support in routers, which still is not widely available across the Internet infrastructure.

Since the native IP multicast has not received widespread deployment, many industrial and research efforts shifted to investigating and deploying the *application level multicast*, where nodes across the Internet act as intermediate routers to efficiently distribute content along a predefined mesh or tree. A growing number of researchers [7, 9, 12, 13, 17, 10, 6, 14] have advocated this alternative approach, where all multicast related functionality, including group management and packet replication, is implemented at end systems. In this architecture, nodes participating in the multicast group self organize themselves into an scalable overlay structure using a distributed protocol. Further, the nodes attempt to optimize the efficiency of the overlay by adapting to changing network conditions and considering the application level requirements.

An interesting extension for the end-system multicast is introduced in [6], where authors, instead of using the end systems as routers forwarding the packets, propose that the end-systems do actively collaborate in informed manner to improve the performance of large file distribution. The main idea is to overcome the limitation of the traditional service models based on tree topologies where the transfer rate to the client is defined by the bandwidth of the bottleneck link of the path from the server. The authors propose to use additional cross-connections between the end-systems to exchange the complementary content these nodes have already received. Assuming that any given pair of end-systems has not received exactly the same content, these cross-connections between the end-systems can be used to "reconcile" the differences in received content in order to reduce the total transfer time.

In our work, we consider a geographically distributed network of servers and a problem of content distribution across it. Our focus is on distributing large size files such as software packages or stored streaming media files (also called as on-demand streaming media). We propose a novel algorithm, called *FastReplica*, for efficient and reliable replication of large files. There are a few basic ideas exploited in *FastReplica*. In order to replicate a large file among n nodes (n is in the range of 10-30 nodes), the original file is partitioned into n subfiles of equal size and each subfile is transferred to a different node in the group (this way, we introduce a "guaranteed", predictable difference in received content as compared to [6]). After that, each node propagates its subfile to the remaining nodes in the group. Thus instead of the typical replication of an entire file to n nodes by using n Internet paths connecting the original node to the replication group, *FastReplica* exploits $n \times n$ diverse Internet paths within the replication group where each path is used for transferring $\frac{1}{n}$-th of the file (in such a way, similarly to [6], we exploit explicitly the additional cross-connections between the end-systems to exchange the complementary content these nodes have already received).

The paper is organized as follows. Section 2 describes additional related work. In Section 3.2, we introduce a core (an induction step) of the algorithm, called *FastReplica in the small*, and demonstrate its work in the small-scale environment, where a set of nodes in a replication set is rather small and limited, e.g. 10 - 30 nodes. In Section 3.3, we perform a preliminary analysis of *FastReplica in the small* and its potential performance benefits, and outline the configurations and network conditions when *FastReplica* may be inefficient. Then in Section 3.4, using *FastReplica in the small* as the induction step, we design a scalable *FastReplica* algorithm which can be used for replication of large files to a large number of nodes. Finally, in Section 3.5, we show how to extend the algorithm for resilience to nodes' failures.

Through experiments on a prototype implementation, we analyze the performance of *FastReplica in the small* in a wide-area testbed in Section 4. For comparison reasons, we introduce *Multiple Unicast* and *Sequential Unicast* schemas. Under *Multiple Unicast*, the source node simultaneously transfers the entire file to all the recipient nodes via concurrent connections. This schema is traditionally used in the small-scale environment. *Sequential Unicast* schema approximates the file distribution under IP multicast. The experiments show that *FastReplica* significantly reduces the file replication time. On average, it outperforms *Multiple Unicast* by $\frac{n}{2}$ times, where n is the number of nodes in the replication set. Additionally, *FastReplica* demonstrates better or comparable performance against file distribution under *Sequential Unicast*.

While we observed significant performance benefits under *FastReplica* in our experiments, these results are sensitive to a system configuration and bandwidth of the paths between the nodes.

Since *FastReplica in the small* represents an induction step of the general *FastReplica* algorithm, these performance results set the basis for performance expectations of *FastReplica in the large*.

2 Additional Related Work

The recent work on content distribution can be largely divided into three categories: (a) infrastructure based content distribution, (b) overlay network based distribution [7, 9, 12, 13, 17, 10, 6, 14], and (c) peer-to-peer content distribution [11, 16, 1, 24].

Our work is directly related to the infrastructure based content distribution network (CDN) (e.g. Akamai), which employs a dedicated set of machines to re-

liably and efficiently distribute content to clients on behalf of the server. While the entire collection of nodes in a CDN setting may be varying, we assume that the set of currently active nodes is known. The sites supported by multiple mirror servers are referred to the same category. Existing research on CDNs and server replication has primarily focused on either techniques for efficient redirection of user requests to appropriate servers or content/server placement strategies for reducing the latency of end-users.

A more recent idea is to access multiple servers in parallel to reduce downloading time or to achieve fault tolerance. Several research papers in this direction exploited the benefits of path diversity between the clients and the site's servers with replicated content. Authors in [20], demonstrate the improved response time observed by the client for a large file download through the dynamic parallel access schema to replicated content at mirror servers. Digital Fountain [4] applies Tornado codes to achieve a reliable data download. Their subsequent work [5] reduces the download times by having a client receive a Tornado encoded file from multiple mirror servers. The target application of their approach is bulk data transfer.

While CDNs were originally intended for static web content, they have been applied for delivery of streaming media as well. Delivering streaming media over the Internet is challenging due to a number of factors such as high bit rates, delay and loss sensitivity. Most of the current work in this direction concentrates on how to improve the media delivery from the edge servers (or mirror servers) to the end clients.

In order to improve streaming media quality, the latest work in this direction [3, 15] proposes streaming video from multiple edge servers (or mirror sites), and in particular, by combining the benefits of multiple description coding (MDC) [2] with Internet path diversity. MDC codes a media stream into multiple complementary descriptions. These descriptions have the property that if either description is received it can be used to decode the baseline quality video, and multiple descriptions can be used to decode improved quality video.

One of the basic assumptions in the research papers referred to above is that the original content is already replicated across the edge (mirror) servers. The goal of our paper is to address the content distribution within this infrastructure (and not to the clients of this infrastructure). In this work, we propose a method to efficiently replicate the content (represented by large files) from a single source to a large number of servers in a scalable and reliable way. We exploit ideas of partitioning the original file and using diverse Internet paths between the recipient nodes to speedup the distribution of an original large file over Internet.

In the paper, we partition the original file in n equal subsequent subfiles and apply *FastReplica* to replicate them. This part of the algorithm can be modified accordingly to the nature of the file. For example, for a media file encoded with MDC, different descriptions can be treated as subfiles, and *FastReplica* can be applied to replicate them. Taking into account the nature of MDC (i.e. that either description received by the recipient node can be used to decode the baseline quality video), the part of the *FastReplica* algorithm dealing with nodes failure can be simplified.

3 *FastReplica* Algorithm

In this Section, we describe the formal problem definition and introduce the new algorithm *FastReplica* for replicating the large files in the Internet environment.

In Section 3.2, we introduce a core (an induction step) of the algorithm, called *FastReplica in the small*, and demonstrate its work in the small-scale environment, where a set of nodes to which a file has to be replicated is rather small and limited, e.g. 10 - 30 nodes.

In Section 3.3, we perform a preliminary analysis of *FastReplica in the small* and its potential performance benefits, and outline the configurations and network conditions when *FastReplica* may be inefficient.

In Section 3.4, we describe the general, scalable algorithm, called *FastReplica in the large*, and demonstrate its work in the large-scale environment, where a set of nodes to which a file has to be replicated can be in a range of hundreds/thousands of nodes.

In Section 3.5, we extend the *FastReplica* algorithm to be able to deal with node failures.

3.1 Problem Statement

We use the following notations:

- Let N_0 be a node which has an original file F and let $Size(F)$ denote the size of file F in bytes;
- Let $R = \{N_1, ..., N_n\}$ be a replication set of nodes.

The problem consists in replicating file F across nodes $N_1, ..., N_n$ while minimizing the overall replication time.

3.2 *FastReplica* in the Small

In this Section, we describe a *core* of *FastReplica* which is directly applicable to a case when a set of recipient nodes $N_1, ..., N_n$ is small, e.g. in a range of 10-30 nodes.

File F is divided in n equal subsequent subfiles:

$$F_1, ..., F_n$$

where $Size(F_i) = \frac{Size(F)}{n}$ bytes for each i: $1 \leq i \leq n$.

Step 1: Distribution Step.

The originator node N_0 opens n concurrent network connections to nodes $N_1, ..., N_n$, and sends to each recipient node N_i $(1 \leq i \leq n)$ the following items:

- a distribution list of nodes $R = \{N_1, ..., N_n\}$ to which subfile F_i has to be sent on the next step;
- subfile F_i.

The activities taking place on the first step of the *FastReplica* algorithm are shown in Figure 1. We will denote this step as a *distribution step.*

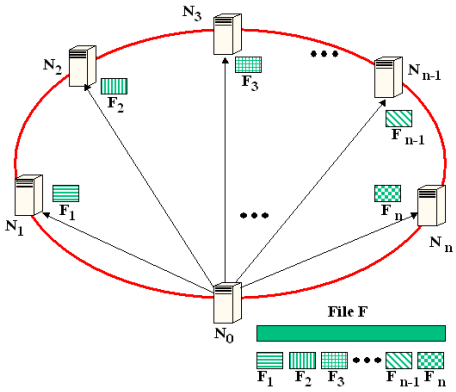

Figure 1: *FastReplica in the small*: distribution step.

Step 2: Collection Step.

After receiving file F_i, node N_i opens $n-1$ concurrent network connections to remaining nodes in the group and send subfile F_i to them as shown in Figure 2 for node N_1.

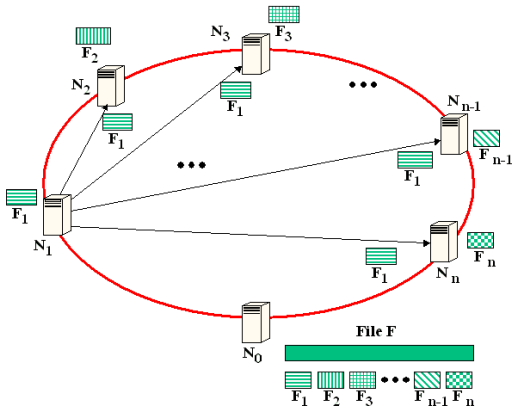

Figure 2: *FastReplica in the small*: a set of outgoing connections of node N_1 at collection step.

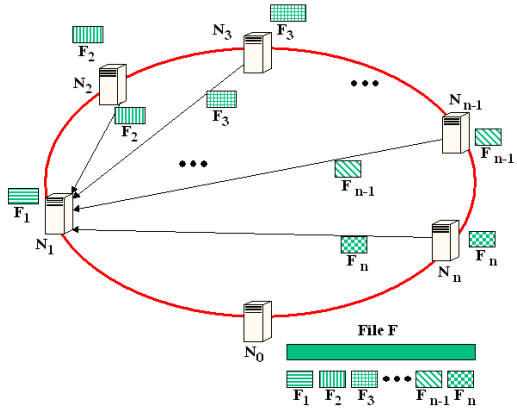

Figure 3: *FastReplica in the small*: a set of incoming connections of node N_1 at collection step.

Similarly, Figure 3 shows the set of incoming concurrent connections to node N_1 from the remaining nodes $N_2, ..., N_n$ transferring the complementary subfiles $F_2, ..., F_n$ during the second logical step of the algorithm. Thus at this step, each node N_i has the following set of network connections:

- there are $n-1$ outgoing connections from node N_i: one connection to each node N_k $(k \neq i)$ for sending the corresponding subfile F_i to node N_k.
- there are $n-1$ incoming connections to node N_i: one connection from each node N_k $(k \neq i)$ for sending the corresponding subfile F_k to node N_i.

Thus at the end of this step, each node receives all subfiles $F_1, ..., F_n$ comprising the entire original file F. We will denote this step as a *collection step.*

In summary, the main idea behind *FastReplica* is that instead of the typical replication of an entire file to n nodes by using n Internet paths connecting the original node to the replication group, the *FastReplica* algorithm exploits $n \times n$ different Internet paths within the replication group where each path is used for transferring $\frac{1}{n}$-th of the file. Thus, the impact of congestion on any particular Internet path participating in the schema is limited for a transfer of $\frac{1}{n}$-th of the file. Additionally, *FastReplica* takes advantage of the upload and download bandwidth of the recipient nodes.

3.3 Preliminary Performance Analysis of *FastReplica* in the Small

Let $Time^i(F)$ denote the transfer time of file F from the original node N_0 to node N_i as measured at node N_i. We use *transfer time* and *replication time* interchangeably in the text. In our study, we consider the following two performance metrics:

- *Average replication time:*

$$Time_{aver} = \frac{1}{n}\sum_{i=1}^{n} Time^i(F)$$

- *Maximum replication time:*

$$Time_{max} = max\{Time^i(F)\}, i \in \{1, \cdots, n\}$$

$Time_{max}$ reflects the time when all the nodes in the replication set receive a copy of the original file, and the primary goal of *FastReplica* is to minimize the maximum replication time. However, we are also interested in understanding the impact of *FastReplica* on the average replication time $Time_{aver}$.

First, let us consider an *idealistic setting*, where nodes $N_1, ..., N_n$ have symmetrical (or nearly symmetrical) incoming and outgoing bandwidth which is typical for CDNs, distributed IDCs, and a distributed enterprise environment. In addition, let nodes $N_0, N_1, ..., N_n$ be homogeneous, and let each node can support k network connections to other nodes at B bytes per second on average.

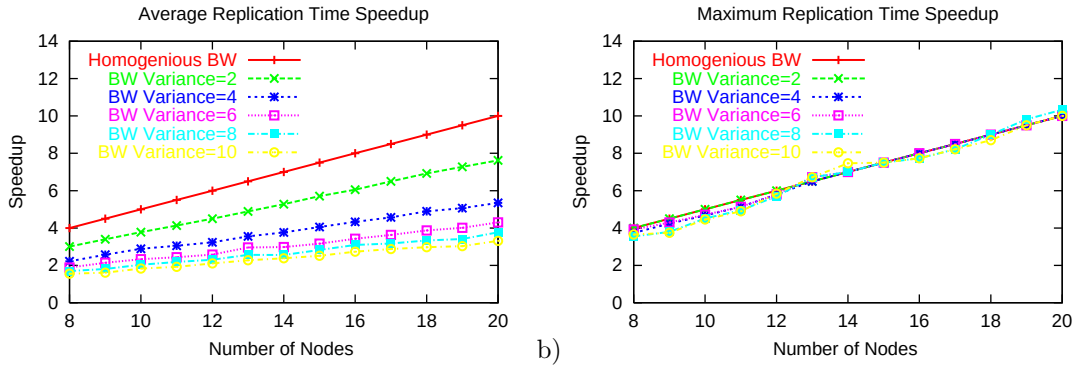

Figure 4: *Uniform-random model*: speedup in file replication time under *FastReplica* vs *Multiple Unicast* for a different number of nodes in replication set for a) average replication time, and b) maximum replication time.

In the idealistic setting, there is no difference between maximum and average replication times. Using the assumption on homogeneity of nodes' bandwidth, we can estimate the transfer time for each concurrent connection i $(1 \leq i \leq n)$ during the *distribution step*:

$$Time_{distr} = \frac{Size(F)}{n \times B} \quad (1)$$

The transfer time at the *collection step* is similar to the time encountered at the first (distribution) step:

$$Time_{collect} = \frac{Size(F)}{n \times B} \quad (2)$$

Thus the overall replication time under *FastReplica in the small* is the following:

$$Time_{FR}^{small} = Time_{distr} + Time_{collect} = 2 \times \frac{Size(F)}{n \times B} \quad (3)$$

Let *Multiple Unicast* denote a schema that transfers the entire file F from the original node N_0 to nodes $N_1, ..., N_n$ by simultaneously using n concurrent network connections. The overall transfer time under *Multiple Unicast* is the following:

$$Time_{MU}^{small} = \frac{Size(F)}{B} \quad (4)$$

Thus, in an idealistic setting, *FastReplica in the small* provides the following speedup of file replication time compared to the *Multiple Unicast* strategy:

$$Replication_Time_Speedup = \frac{Time_{FR}^{small}}{Time_{MU}^{small}} = \frac{n}{2} \quad (5)$$

While the comparison of *FastReplica* and *Multiple Unicast* in the idealistic environment gives insights into why the new algorithm may provide significant performance benefits for replication of the large files, the bandwidth conditions in the realistic setting could be very different from the idealistic assumptions. Due to changing network conditions, even the same link might have a different available bandwidth when measured at different times. Let us analyze how *FastReplica* performs when network paths participating in the transfers have a different available bandwidth.

Let BW denote a *bandwidth matrix*, where $BW[i][j]$ reflects the available bandwidth of the path from node N_i to node N_j as measured at some time T, and let Var be the ratio of maximum to minimum available bandwidth along the paths participating in the file transfers. We call Var a *bandwidth variation*.

In our analysis, we consider the bandwidth matrix BW to be populated in the following way:

$$BW[i][j] = B \times random(1, Var),$$

where function $random(1, Var)$ returns a random integer var: $1 \leq var \leq Var$.

While it is still a simplistic model, it helps to reflect a realistic situation, where the available bandwidth of different links can be significantly different. We will call this model a *uniform-random model*. To perform a sensitivity analysis of how the *FastReplica* performance depends on a bandwidth variation of participating paths, we experimented with a range of different values for Var between 1 and 10. When $Var = 1$, it is the *idealistic setting*, discussed above, where all of the paths are homogeneous and have the same bandwidth B (i.e. no variation in bandwidth). When $Var = 10$, the network paths between the nodes have highly variable available bandwidth with a possible difference of up to 10 times.

Using the *uniform-random* model and its bandwidth matrix BW, we compute the average and maximum file replication times under *FastReplica* and *Multiple Unicast* methods for a different number of nodes in the replication set, and derive the relative speedup of the file replication time under *FastReplica* compared to the replication time under the *Multiple Unicast* strategy. For each value of Var, we repeated the experiments multiple times, where the bandwidth matrix BW is populated by using the random number generator with different seeds.

Figure 4 a) shows the relative average replication time speedup under *FastReplica in the small* compared to *Multiple Unicast* in the *uniform-random* model. For

Var=2, the average replication time for *8* nodes under *FastReplica* is *3* times better compared to *Multiple Unicast*, and for *20* nodes, it is *8* times better. While the performance benefits of *FastReplica* against *Multiple Unicast* are decreasing for higher variation of bandwidth of participating paths, *FastReplica* still remains quite efficient, with performance benefits converging to a practically fixed ratio for $Var > 4$.

Figure 4 b) shows the relative maximum replication time speedup under *FastReplica in the small* compared to *Multiple Unicast* in the *uniform-random* model. We can observe that, independent of the values of bandwidth variation, the maximum replication time under *FastReplica* for n nodes is $\frac{n}{2}$ times better compared to the maximum replication time under *Multiple Unicast*.

It can be explained in the following way:

- *Multiple Unicast*: The maximum replication time is defined by the entire file transfer time over the path with the worst available bandwidth among the paths connecting N_0 and N_i, $1 \leq i \leq n$.
- *FastReplica*: Figure 5 shows the set of paths participating in the file transfer from node N_0 to node N_1 under the *FastReplica* algorithm (we use N_1 as a representative of the recipient nodes).

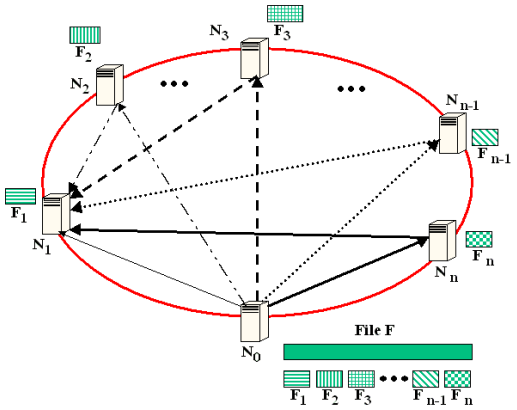

Figure 5: *FastReplica in the small*: a set of paths used in file F replication from node N_0 to node N_1.

The replication time observed at node N_1 is defined by the maximum transfer time of $\frac{1}{n}$-th of the file over either:

- the path from N_0 to N_1, or
- the path with the worst overall available bandwidth consisting of two subpaths:
  - the subpath from N_0 to N_j and
  - the subpath from N_j to N_1,

  for some $j : 1 \leq j \leq n$.
  In the considered *uniform-random* model, a worst case scenario is when both subpaths have a minimal bandwidth, and since each path is used for transferring $\frac{1}{n}$-th of the entire file, this would lead to $\frac{n}{2}$ times latency improvement under *FastReplica* compared to the maximum replication time under *Multiple Unicast*.

Now, let us consider a special, somewhat artificial example, which aims to provide an additional insight into the possible performance outcomes under *FastReplica* when the values of bandwidth matrix BW are significantly skewed.

Let N_0 be the origin node, and $N_1, ..., N_{10}$ be the recipient nodes, and the bandwidth between the nodes be defined by the following matrix:

$$BW(i,j) = \begin{cases} \frac{1}{10} \times B & \text{if i=0, j=1} \\ B & \text{if i=0, } 2 \leq j \leq 10 \\ \frac{1}{10} \times B & \text{if } 1 \leq i,j \leq 10 \end{cases} \quad (6)$$

In other words, the origin node N_0 has a limited bandwidth of $\frac{1}{10} \times B$ to node N_1, while the bandwidth from N_0 to the rest of the recipient nodes $N_2, ..., N_{10}$ is equal to B. In addition, the cross-bandwidth between the nodes $N_1, ..., N_{10}$ is also very limited, such that any pair N_i and N_j is connected via a path with available bandwidth of $\frac{1}{10} \times B$.

At a glance, it seems that *FastReplica* might perform badly in this configuration because the additional cross-bandwidth between the recipient nodes $N_1, ..., N_{10}$ is so poor relative to the bandwidth available between the origin node N_0 and the recipient nodes $N_2, ..., N_{10}$. Let us compute the average and maximum replication times for this configuration under *Multiple Unicast* and *FastReplica* strategies.

- *Multiple Unicast*:

$$Time_{aver} = \frac{19 \times Size(F)}{10 \times B}, Time_{max} = \frac{10 \times Size(F)}{B}.$$

- *FastReplica*:

$$Time_{aver} = \frac{191 \times Size(F)}{100 \times B}, Time_{max} = \frac{2 \times Size(F)}{B}.$$

The maximum replication time in this configuration is 5 times better under *FastReplica* than under *Multiple Unicast*. In *FastReplica*, any path between the nodes is used to transfer only $\frac{1}{n}$-th of the entire file. Thus, the paths with poor bandwidth are used for much shorter transfers which leads to a significant improvement in maximum replication time. However, the average replication time in this example is not improved under *FastReplica* compared to *Multiple Unicast*. The reason for this is that the high bandwidth paths in this configuration are used similarly: to transfer only $\frac{1}{n}$-th of the entire file, and during the collection step of the *FastReplica* algorithm, the transfers of complementary $\frac{1}{n}$-th size subfiles within the replication group are performed over poor bandwidth paths. Thus, in certain cases, like considered above, *FastReplica* may provide significant improvements in maximum replication time, but may not improve the average replication time.

The analysis considered in this section outlines the conditions when *FastReplica* is expected to perform well,

providing the essential performance benefits. Similar reasoning can be applied to derive the situations when *FastReplica* might be inefficient. For example, let us slightly modify the previous example. Let the bandwidth matrix BW be defined in the following way:

$$BW(i,j) = \begin{cases} B & \text{if i=0, } 1 \le j \le 10 \\ \frac{1}{10} \times B & \text{if } 1 \le i,j \le 10 \end{cases} \quad (7)$$

In this configuration, the bandwidth from the origin node N_0 to the rest of the recipient nodes $N_1, ..., N_{10}$ is equal to B, while the cross-bandwidth between the nodes $N_1, ..., N_{10}$ is very limited: any pair N_i and N_j is connected via a path with available bandwidth of $\frac{1}{10} \times B$. The average and maximum replication times for this configuration under *Multiple Unicast* and *FastReplica* strategies can be computed as follows:

- *Multiple Unicast*: $Time_{aver} = Time_{max} = \frac{Size(F)}{B}$.
- *FastReplica*: $Time_{aver} = Time_{max} = \frac{11 \times Size(F)}{10 \times B}$.

Thus in this configuration, *FastReplica* does not provide any performance benefits.

In a general case, if there is a node N_k in the replication set such that most of the paths between N_k and the rest of the nodes have a very limited available bandwidth (say, n times worse than the minimal available bandwidth of the paths connecting N_0 and N_i, $1 \le i \le n$) then the performance of *FastReplica* during the second (collection) step is impacted by the poor bandwidth of the paths between N_k and N_i, $1 \le i \le n$, and *FastReplica* will not provide expected performance benefits. Note, that n (the number of nodes in the replication group) plays a very important role here: a larger value of n provides a higher "safety" level for *FastReplica* efficiency. A larger value of n helps to offset a higher difference in bandwidth between the available bandwidth within the replication group and the available bandwidth from the original node to the nodes in the replication group.

To apply *FastReplica* efficiently, the preliminary bandwidth estimates are useful. These bandwidth estimates are also essential for correct clustering of the appropriate nodes into the replication subgroups in *FastReplica in the large* discussed in the next section.

3.4 *FastReplica* in the Large

In this Section, we generalize *FastReplica in the small* to a case where a set of nodes to which a file has to be replicated can be in the range of hundreds/thousands of nodes.

Let k be a number of network connections chosen for concurrent transfers between a single node and multiple receiving nodes (i.e. k limits the number of nodes in the group for *Multiple Unicast* or *FastReplica* strategies). An appropriate value of k can be experimentally determined via probing. Heterogeneous nodes might be capable of supporting a different number of connections. Let k be the number of connections suitable for most of the nodes in the overall replication set.

A natural way to scale *FastReplica in the small* to a large number of nodes is:

- partition the original set of nodes into replication groups, each consisting of k nodes;
- apply *FastReplica in the small* iteratively: first, replicate the original file F to a group of k nodes, and then use these k nodes as the origin nodes with file F to repeat the same procedure to a new groups of nodes, etc.

Schematically, this procedure is shown in Figure 6, where circles represent the nodes, and boxes represent the replication groups. The arrows, connecting one node with a set of other nodes, reflect the origin node and the recipient nodes, involved in communications on a particular iteration of the algorithm.

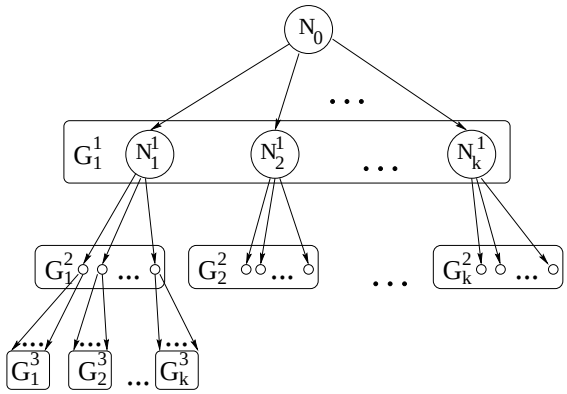

Figure 6: *FastReplica in the large*: iterative replication process.

At the first iteration, node N_0 replicates file F to group G_1^1, consisting of k nodes, by using the *FastReplica in the small* algorithm.

At the second iteration, each node $N_i^1 (1 \le i \le k)$ of group G_1^1 can serve as the origin node propagating file F to another group G_i^2.

Thus in two iterations, file F can be replicated to $k \times k$ nodes. Correspondingly, in three iterations, file F can be replicated to $k \times k \times k$ nodes.

The general *FastReplica* algorithm is based on the reasoning described above. Let the problem consist in replicating file F across nodes $N_1, ..., N_n$ and let $\frac{n}{k} = m$. Then all the nodes are partitioned into m groups:

$$G^1, G^2, ..., G^m$$

where each group has k nodes.

Any number m can be represented as

$$m = c_1 \times k^{i_1} + c_2 \times k^{i_2} + ... + c_j \times k^{i_j} \quad (8)$$

where $i_1 > i_2 > ... > i_j \ge 0$ and $0 < c_1, ..., c_j < k$. Practically, it is a k-ary representation of a number m.

This representation defines the rules for constructing the tree structure similar to the one shown in Figure 6. In particular, the height of such a tree is i_1+1, and it defines the number of iterations in the general *FastReplica* algorithm.

From this representation, the rules for constructing the corresponding distribution lists of nodes are straightforward. We omit the technical details of the distribution lists construction in order to keep the description of the overall algorithm concise.

If the targeted number n of nodes for a file replication is not a multiple of k, i.e.

$$\frac{n}{k} = m + r$$

where $r < k$, then there is one "incomplete" group $\hat{G}$ with r nodes in it. The best way to deal with this group is to arrange it to be a leaf-group in the shortest subtree. Let $G' = \{N'_1, ..., N'_k\}$ be a replication group in the shortest subtree.

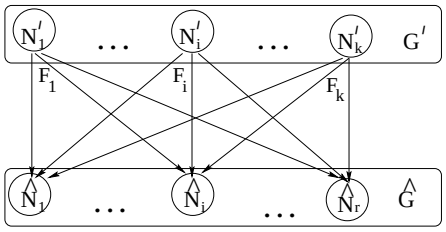

Figure 7: Communications between the nodes of regular replication group G' and incomplete replication group $\hat{G}$: *special step.*

The communications between groups G' and $\hat{G}$ follow a slightly different file exchange protocol. All the nodes in G' have already received all subfiles $F_1, ..., F_n$ comprising the entire original file F. Each node N'_i of group G' opens r concurrent network connections to all r nodes of group $\hat{G}$ for transferring its subfile F_i as shown in Figure 7. In this way, at the end of this step, each node of group $\hat{G}$ has all subfiles $F_1, ..., F_k$ of the original file F. We will denote this step as a *special step.*

Example. Let $k = 10$. How many algorithm iterations are required to replicate the original file to 1000 nodes? Using Formula (8) above, we derive the following representation for 1000 nodes:

$$1000 = 10 \times 10^2$$

Thus, in three algorithm iterations ($10 \times 10 \times 10$), the original file can be replicated among all 1000 nodes. At each iteration, the replication process follows *FastReplica in the small*, i.e. the iteration consists of 2 steps, each used for transferring the $\frac{1}{k}$-th portion of the original file F.

Let *Multiple Unicast* follow a similar recursive replication tree as the one defined above in general *FastReplica* and shown in Figure 6, with the only difference being that communications between the origin nodes and the recipient nodes follow the *Multiple Unicast* schema, i.e. the origin node transfers the entire file F to the corresponding recipient nodes by simultaneously using k concurrent network connections. Thus, in three algorithm iterations, by using *Multiple Unicast* recursively, the original file can be replicated among 1000 nodes.

3.5 Reliable *FastReplica* Algorithm

In this Section, we extend the *FastReplica* algorithm to be able to deal with node failures. The basic algorithm presented in Sections 3.2, 3.4 is sensitive to node failures. For example, if node N_1 fails during either transfer shown in Figures 1, 2 then this event may impact all nodes $N_2, ..., N_n$ in the group because each node depends on node N_1 to receive subfile F_1. In the described scenario, node N_1 is acting as a recipient node in the replication set. If a node fails when it acts as the origin node, e.g. node N_1^1 in Figure 6, this failure impacts all of the replication groups in the replication subtree rooted in node N_1^1.

The reliable *FastReplica* algorithm proposed below efficiently deals with node failures by making the local repair decision within the particular group of nodes. It keeps the main structure of the *FastReplica* algorithm practically unchanged while adding the desired property of resilience to node failures.

In reliable *FastReplica*, the nodes of each group are exchanging the heartbeat messages with their origin node. The heartbeat messages from nodes to their origin node are augmented with additional information on the corresponding algorithm step and group (list) of nodes to which the nodes currently perform their transfers.

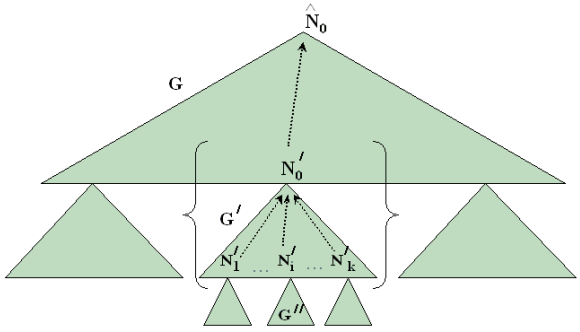

Figure 8: Heartbeat group: the recipient nodes in $G' = \{N'_1, ..., N'_k\}$ send heartbeat messages to the origin node N'_0.

In Figure 8, the nodes $N'_1, ..., N'_k$ of group G' form the heartbeat group with their origin node N'_0. Each node N'_i sends to N'_0 the heartbeat messages with additional information on node state in the replication process. Similarly, node N'_0 belongs to group G with the corresponding origin node $\hat{N}_0$. Thus node N'_0 sends the

heartbeat messages and its node state to $\hat{N}_0$.

There are different repair procedures depending on whether a failed node was acting as a recipient node, e.g. node N'_i in replication set G', or a failed node was acting as an origin node, e.g. N'_0 for replication set G'.

- If node N'_i fails while acting as a recipient node in replication set G' during the *distribution step* then the communication pattern is similar to the pattern shown in Figure 1. In this case, node N'_0 is aware of the node N'_i failure. Node N'_0 performs the following repair step: it uses $k-1$ already opened connections to the rest of the nodes in group G' to send the missing F_i file to each node in the group as shown in Figure 9.

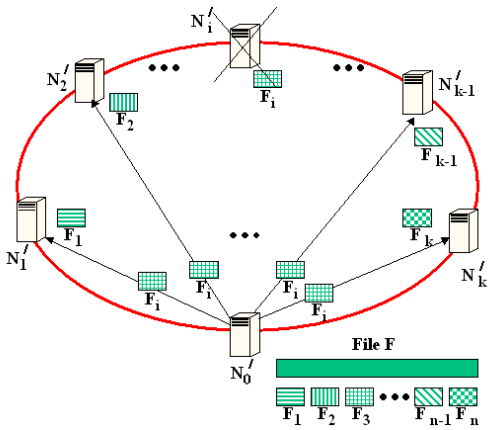

Figure 9: Repair procedure for node N'_i failed during *distribution step*.

  In this way, each node in group G' receives all of the subfiles of the original file F.

  Additionally, node N'_0 acts as a "substitute" for the failed node N'_i in the next algorithm step. If node N'_i was supposed to serve as the origin node to group G'' for the next algorithm iteration, then node N'_0 acts as the origin node to group G'' for this iteration.

- If node N'_i fails while acting as a recipient node in replication set G' during the *collection step* then the communication pattern is similar to the pattern shown in Figure 2. Using the heartbeat messages, the failure of node N'_i is detected by node N'_0. Node N'_0 performs the following repair step: it opens connections to the impacted nodes in group G' to send missing file F_i (similar to the repair step shown in Figure 9). In this way, each node in group G' receives all of the subfiles of the original file F.

  Analogously, node N'_0 acts as a substitute for the failed node N'_i in the next algorithm step.

- If node N'_0 fails while acting as the origin node for replication group G' during the distribution step then replication group G' should be "reattached" to a higher-level origin node. Let $\hat{N}_0$ be the corresponding origin node for N'_0 from the previous iteration step as shown in Figure 8. From heartbeat messages, node $\hat{N}_0$ detects node N'_i failure. Node $\hat{N}_0$ analyzes what was the node N'_0 state in the replication process preceding its failure. Then node $\hat{N}_0$ acts as a replacement for N'_0: it opens connections to the impacted nodes in group G' to send corresponding missing files. Additionally, $\hat{N}_0$ updates every node in G' about the change of the origin node (for future exchange of heartbeat messages).

Reliable *FastReplica*, described above, aims to minimize the impact of node failures by making the local repair decision within the particular group of nodes. These groups are relatively small, e.g. 10-30 nodes. Each group has the origin node (with the original file for replication) and the recipient nodes. The number of heartbit messages in such a group is very small because only the recipient nodes send heart-bit messages to their origin node, and there are no heart-bit messages between the recipient nodes. This structure significantly simplifies the protocol. Proposed failure mechanism easily handles a single node failure within the group with minimal performance penalty. The main structure of the *FastReplica* algorithm is practically unchanged during the repair steps.

4 Performance Evaluation

We outlined the potential performance benefits of *FastReplica in the small* in Section 3.3. The goal of this section is to analyze the *FastReplica* performance in the real Internet environment. Through experiments on a prototype implementation, we will demonstrate the efficiency of *FastReplica in the small* in a wide-area testbed. Since *FastReplica in the small* defines the iteration step in the general algorithm, these results will set the basis for performance expectations of *FastReplica in the large*.

Using the generous help of summer interns at HPLabs, we built an experimental testbed with 9 nodes. Table 1 and Figure 10 show the 9 hosts participating in our experiments and their geographic locations.

N_0	hp.com	Palo Alto, CA
N_1	utexas.edu	Austin, TX
N_2	umich.edu	Ann Arbor, MI
N_3	gatech.edu	Atlanta, GA
N_4	duke.edu	Durham, NC
N_5	uci.edu	Irvine, CA
N_6	berkeley.edu	Berkeley, CA
N_7	mit.edu	Cambridge, MA
N_8	uiuc.edu	Urbana-Champaign, IL

Table 1: Participating nodes.

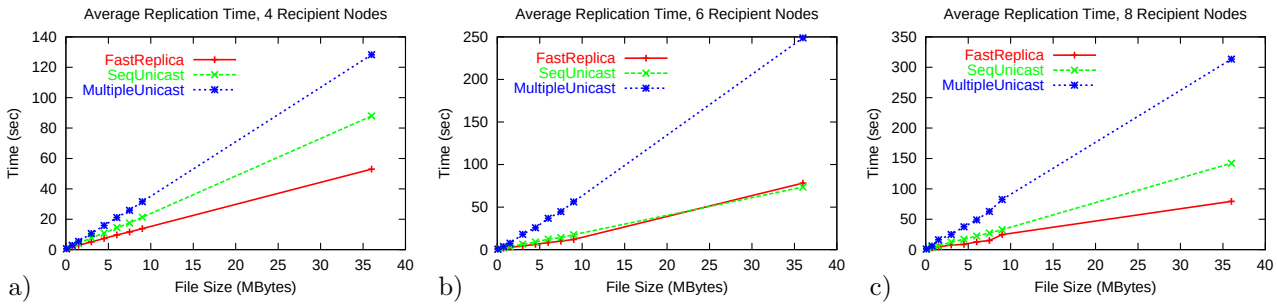

Figure 11: Average replication time for files of different size and a different number of nodes in replication set a) 4 receivers, b) 6 receivers, c) 8 receivers.

The source node is N_0 and is located at the HP site, while the nodes-receivers are at different university sites. In order to perform the sensitivity analysis, we vary the number of participating hosts: in experiments with k participating hosts in replication set, the receivers are $N_1, ..., N_k$ ordered as shown in Table 1.

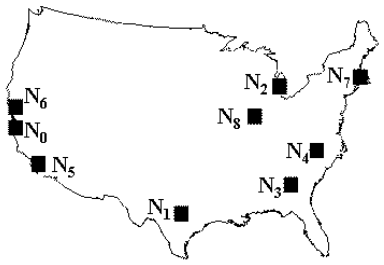

Figure 10: Geographic locations of hosts.

Using the experimental testbed, we compare the following distribution schemes:

- *FastReplica in the small:* we used the *FastReplica* algorithm designed for small, limited number of nodes and introduced in Section 3.2.

- *Sequential Unicast:* this scheme approximates the "best possible overlay tree" for the entire set of group members. For the evaluation, we use the *Sequential Unicast test* which measures the file transfer time from the source to each recipient *independently* (i.e. in the absence of other recipients). Note that *Sequential Unicast* is not a feasible overlay, but a hypothetical construction used for comparison purposes. The measurements under *Sequential Unicast* approximate the file distribution using IP multicast.

- *Multiple Unicast:* under this scheme, the original node simultaneously transfers the entire file to all the recipient nodes by using the concurrent connections. Assuming an infinite bandwidth at the original node, this scheme can be considered as a feasible solution for the above "best possible overlay tree".

The experiments are conducted at an application level. Ideally, the transfer time at each recipient node should be measured from the beginning of the transfer at the source node to the completion of the file download at the recipient node. However, due to clock synchronization problems at different nodes, we measure the file transfer time at each recipient node from the beginning of file download to the end of file download at the corresponding recipient node. Since we are interested in large file transfers, the omission of one-way latency of the first packet from the source to the recipient cannot impact the accuracy of the results.

In our study, we consider the two performance metrics introduced in Section 3.3: *average replication time* and *maximum replication time*.

To analyze the efficiency of *FastReplica*, we performed its sensitivity analysis for replication of different size files and across different numbers of nodes in the replication set. We experimented with 9 file sizes: 80 Kbytes, 750 Kbytes, 1.5 MBytes, 3 MBytes, 4.5 MBytes, 6 MBytes, 7.5 MBytes, 9 MBytes, and 36 MBytes, and varied the number of nodes in the replication set from 2 to 8. When running experiments with different parameters and strategies, the experiments for the same file size were clustered in time as closely as possible to eliminate biases due to short time scale changes in network and system conditions. In order to eliminate the biases due to longer time scale changes in network and system conditions, we performed the same set of experiments at different times of the day. Each point in the results is averaged over 10 different runs which were performed over a 10 day period.

Figure 11 shows the average file replication time for experiments with 4, 6, and 8 recipient nodes in the replication set and files of different sizes. For file sizes larger than 80 Kbytes, *FastReplica* significantly outperforms *Multiple Unicast*. The replication time under *FastReplica* is 2-4 times better than under *Multiple Unicast*.

Additionally, in most experiments, *FastReplica* outperforms *Sequential Unicast*, which approximates the file replication with IP multicast. The explanation is that when *Sequential Unicast* replicates file F to n nodes, it uses n Internet paths connecting the source

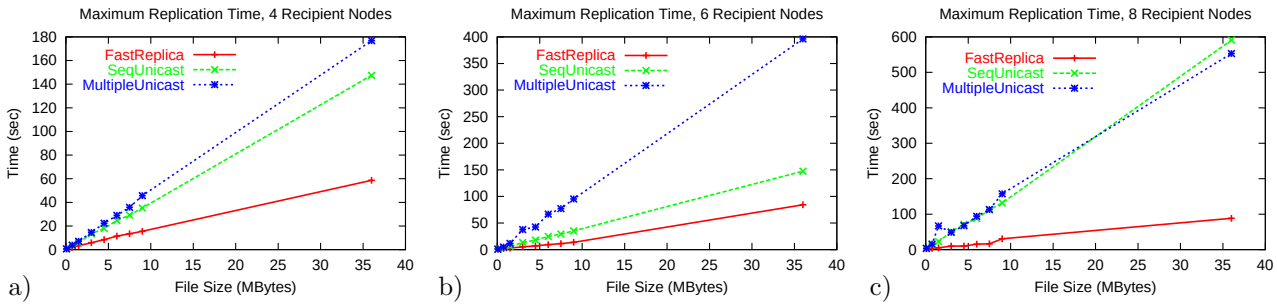

Figure 12: Maximum replication time for files of different size and a different number of nodes in replication set a) 4 receivers, b) 6 receivers, c) 8 receivers.

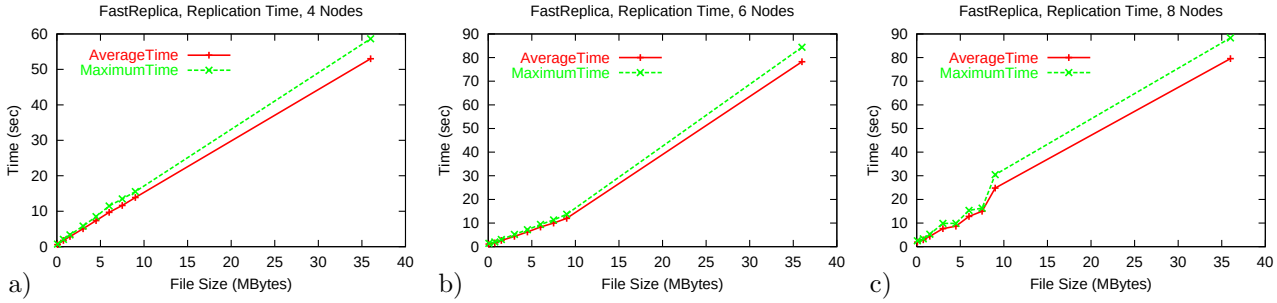

Figure 13: *FastReplica*: average vs maximum replication time for a different number of nodes in replication set a) 4 receivers, b) 6 receivers, c) 8 receivers.

nodes to the recipient nodes (while sending only one packet over each common link in those paths). Thus the overall performance is defined by the end-to-end properties of the n paths. Congestion in any of those paths impacts the overall performance of the *Sequential Unicast*. *FastReplica* uses the same n paths between the source and recipient nodes to transfer only $\frac{1}{n}$-th of file F. *FastReplica* takes advantage of using the additional $(n-1) \times n$ paths between the nodes in the replication set, and each of those paths is used for sending $\frac{1}{n}$-th of file F. Thus, the congestion in any of those paths impacts *FastReplica* performance for transfer of only the $\frac{1}{n}$-th of file F.

While the average replication time provides an interesting metric for distribution strategy characterization, the metric representing the maximum replication time is critical, because it reflects the worst case of the replication time among the recipient nodes. Figure 12 shows the maximum replication time for experiments with 4, 6, and 8 recipient nodes in a replication set and files of different sizes. The maximum replication times under *Multiple Unicast*, as well as *Sequential Unicast*, are much higher than the corresponding average times for these strategies. For a case of 8 nodes in the replication set, the maximum times under *Multiple Unicast* and *Sequential Unicast* are almost 2 times higher than the corresponding average times. The reason is that there is a very limited bandwidth on the path from the source node N_0 to the recipient node N_8. The performance of this path is practically the same for both *Multiple Unicast* and *Sequential Unicast*. This path defines the worst (maximum) replication time among all the recipient nodes in the set. Since *FastReplica* uses this path to transfer only $\frac{1}{n}$-th of file F, this "bad" path has a very limited impact on maximum replication time and overall performance of *FastReplica*.

Figure 13 shows how close the average and maximum replication times under *FastReplica* are. These results demonstrate the robustness and predictability of performance results under the new strategy.

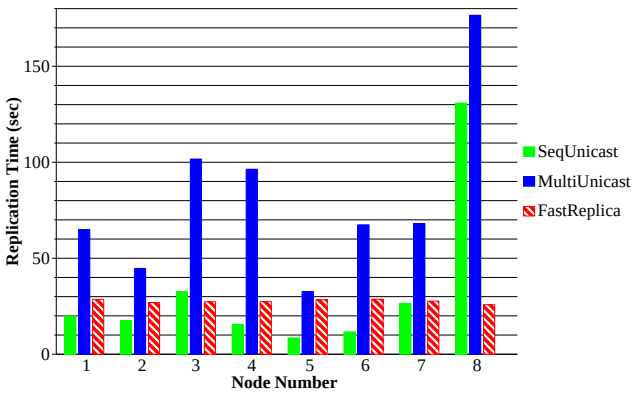

Figure 14: Replication time measured by individual receiving nodes for 9 MB file and 8 nodes in replication set.

Figure 14 shows the average replication time measured by the different, individual recipient nodes for a 9 MB file and 8 nodes in the replication set (the other graphs for different file sizes and a different number

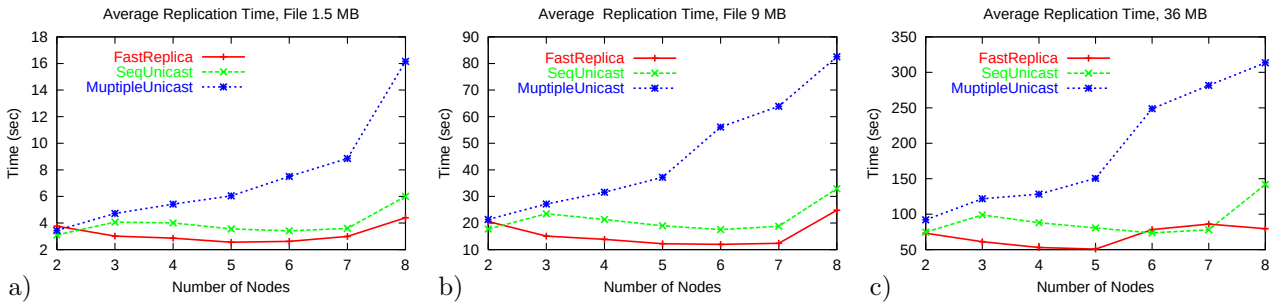

Figure 15: Average file replication time for a different number of nodes in replication set and a) File size of 1.5 MB, b) File size of 9 MB, c) File size of 36 MB.

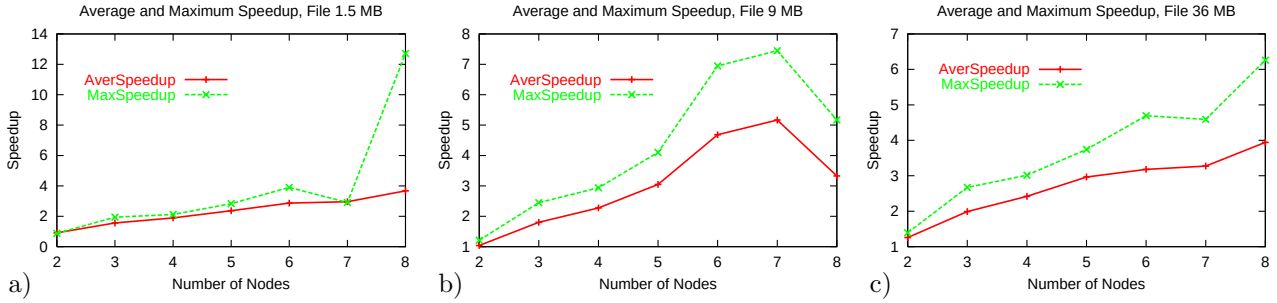

Figure 16: Speedup in average and maximum file replication time under *FastReplica* vs *Multiple Unicast* for a different number of nodes in replication set and a) 1.5 MB file, b) 9 MB file, c) 36 MB file.

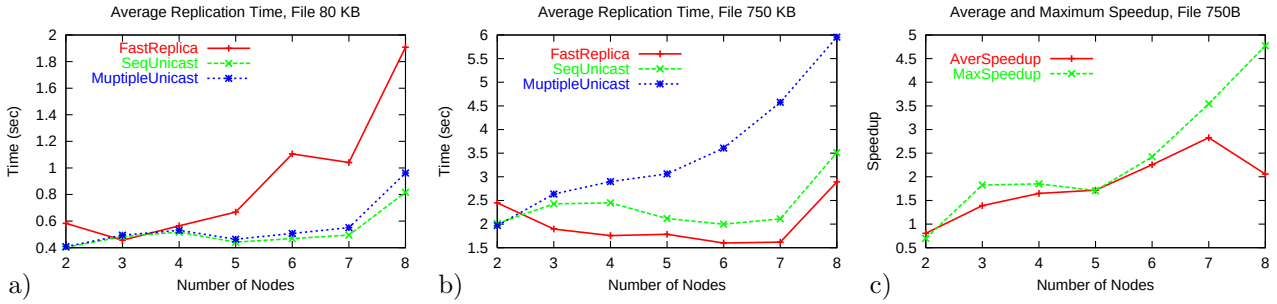

Figure 17: Average file replication time for a different number of nodes in replication set and a) File size of 80 KB, b) File size of 750 KB, c) Speedup in file replication time under *FastReplica* vs *Multiple Unicast* for 750 KB file.

of nodes in the replication set reflect similar trends). There is a high variability of replication time under *Multiple Unicast* and *Sequential Unicast*. This is somewhat expected because the file replication times at the individual nodes highly depend on the available bandwidth of the path connecting the source and receiver node. The limited bandwidth of the path between the original node N_0 and the receiver node N_8 can be observed from these measurements, and it severely impacts the overall performance of both *Multiple Unicast* and *Sequential Unicast*. The file replication times under *FastReplica* across different nodes in the replication set are much more stable and predictable since each node performance is defined by the bandwidth of n paths, each transferring $\frac{1}{n}$-th of the original file F.

Figure 15 shows the average replication time for files of 1.5 MB, 9 MB, and 36 MB for a different number of nodes in the replication set. While *Multiple Unicast* shows a growing replication time for an increasing number of nodes in the replication set, *FastReplica* and *Sequential Unicast* demonstrate good scalability for replication sets of different sizes. Additionally, *FastReplica* consistently outperforms *Sequential Unicast* for most of the points.

Figure 16 shows the average and maximum speedup of file replication time under proposed *FastReplica in the small* relative to the replication time of *Multiple Unicast* for files of 1.5 MB, 9 MB, and 36 MB, and a different number of nodes in the replication set. The results consistently show the significant speedup both in average and maximum replication times across considered different file sizes.

Finally, Figures 17 a) and b) show the average replication time for 80 KB and 750 KB files and a different number of nodes in the replication set. The files of 80 KB and 750 KB are the smallest ones used in our

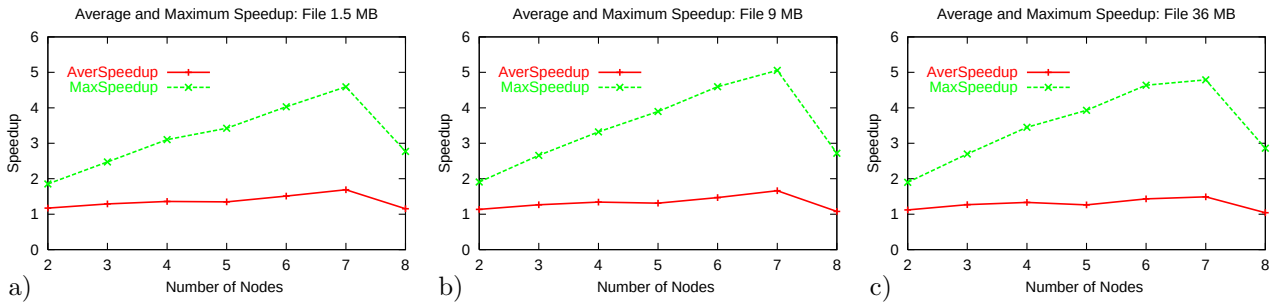

Figure 18: Different configuration with N_1 (*utexas.edu*) being the origin node: speedup in average and maximum replication times under *FastReplica* vs *Multiple Unicast* for a different number of nodes in replication set and a) 1.5 MB file, b) 9 MB file, c) 36 MB file.

experiments. For the 80 KB file, *FastReplica* is not efficient, and the replication time (both average and maximum) is higher than under *Sequential Unicast* and *Multiple Unicast*. For the 750 KB file, the replication time under *FastReplica* is better than under *Sequential Unicast* and *Multiple Unicast* strategies, and the average and maximum speedup shown in Figure 17 c) is again significant. These results help to outline the "border" parameters for new strategy usage: in our case study, *FastReplica* works most efficiently for replicating files larger than 0.5 MB. For $n > 8$, the "border" file size, where *FastReplica* works most efficiently, may increase correspondingly.

The results presented in Figure 16 show the significant speedup both in average and maximum replication times under the *FastReplica* strategy. The additional analysis reveals that the available bandwidth of the paths between the origin node N_0 (*hp.com*) and nodes $N_1, ..., N_7$ (universities' machines) is significantly lower than the cross bandwidth between nodes $N_1, ..., N_7$. only node N_8 has a limited incoming bandwidth from all the nodes $N_0, N_1, ..., N_7$, while the outgoing bandwidth from node N_8 to $N_1, ..., N_7$ is again significantly higher. In such a configuration, *FastReplica* utilizes the abundance of additional available bandwidth between the replication nodes in the most efficient way to produce the spectacular results.

It is interesting to see how *FastReplica* would perform when a different node with high bandwidth paths to the rest of the nodes is used as the origin node. We changed the configuration and made node N_1 (*utexas.edu*) to be the origin node, and rerun the experiments again.

Figure 18 shows the average and maximum speedup of file replication time under the proposed *FastReplica in the small* relative to the replication time of *Multiple Unicast* for files of 1.5 MB, 9 MB, and 36 MB, and a different number of nodes in the replication set in the new configuration.

In the new configuration, the average replication times under *FastReplica* and *Multiple Unicast* are similar, but the maximum replication time under *FastReplica* is still significantly better than the maximum replication time under *Multiple Unicast*.

The bandwidth analysis reveals that node *utexas.edu* is connected to the rest of the nodes via high bandwidth paths with low bandwidth variation across these paths. Our analysis in Section 3.3 with a specially designed example, where the bandwidth matrix BW is defined by equations (6), demonstrates that when the cross bandwidth between some replication nodes is significantly lower than the bandwidth of the original paths from N_0 to the recipient nodes $N_1, ..., N_8$ then *FastReplica* improves the maximum replication time but may have no significant improvement in average replication time.

5 Conclusion

In recent years, the Web and Internet services have moved from an architecture where data objects are located at a single origin server or site to the an architecture where objects are replicated across multiple, geographically distributed servers. Client requests for content are redirected to a best-suited replica rather than the origin server. For large files, the replication process across this distributed network of servers is a challenging and resource-intensive problem on its own.

In this work, we introduce *FastReplica* for efficient and reliable replication of large files in the Internet environment. *FastReplica* partitions an original file into a set of subfiles and uses a diversity of Internet paths among the receiving nodes to propagate the subfiles within the replication set in order to speedup the overall download time for the original content. We scale the algorithm by clustering the nodes in a set of replication groups, and by arranging efficient group communications among them, i.e. by building the overlay tree on top of those groups.

Through experiments on a prototype implementation, we demonstrate the efficiency of *FastReplica in the small* in a wide-area testbed. Since *FastReplica in the small* defines the iteration step in the general algorithm, these performance results set the basis for performance expectations of *FastReplica in the large*.

The interesting and important issues for future re-

search are "how to better cluster the nodes in replication groups?" and "how to build an efficient overlay tree on top of those groups?" Recent research [19, 14, 22] shows that the large-scale Internet application could benefit from incorporating IP-level topological information in the construction of the overlay to significantly improve overlay performance. In [19], a new distributed technique is introduced where the nodes partition themselves into bins in such a way that nodes within a given bin are relatively close to one another in terms of network latency. It might be an interesting technique for clustering "close" nodes into replication groups in *FastReplica*.

To analyze and validate future optimization for *FastReplica*, a large-scale Internet environment or testbed is needed. In recent work [23], authors propose ModelNet as a comprehensive Internet emulation environment to evaluate Internet-scale distributed systems. A new initiative within the research community around PlanetLab [18] is aiming to build a global testbed for developing and accessing new network services. The introduction of such environments and large-scale testbeds will help to support interesting scalability experiments in the near future.

Acknowledgements: We would like to thank HPLabs summer interns who helped us to build an experimental wide-area testbed from their university machines: Yun Fu, Weidong Cui, Taehyun Kim, Kevin Fu, Zhiheng Wang, Shiva Chetan, Xiaoping Wei, and Jehan Wickramasuriya. Their help is highly appreciated.

Authors also would like to thank John Apostolopoulos for motivating discussions on multiple descriptions for streaming media and John Sontag for his active support of this work.

We would like to thank the anonymous referees for useful remarks and insightful questions, and our shepherd Srinivasan Seshan for constructive suggestions to improve the content and presentation of the paper.

References

[1] Allcast *http://www.allcast.com/*

[2] J. Apostolopoulos. Reliable video communication over lossy packet network using multiple state encoding and path diversity. *Proc. of Visual Communications and Image Processing (VCIP)*, January, 2001.

[3] J. Apostolopoulos, T. Wong, W. Tan, and S. Wee. On multiple description streaming with content delivery networks. *Proc. of IEEE Infocom*, 2002.

[4] J. Byers, M. Luby, M. Mitzenmacher, A. Rege. A Digital Fountain approach to reliable distribution of bulk data. *Proc. of ACM SIGCOMM*, 1998.

[5] J. Byers, M. Luby, M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using Tornado codes to speedup downloads. *Proc. of IEEE Infocom*, 1999.

[6] J. Byers, J. Considine, M. Mitzenmacher, S. Rost. Informed content delivery across adaptive overlay networks. *Proc. of ACM SIGCOMM*, 2002.

[7] Y. Chawathe. Scattercast: an architecture for Internet broadcast distribution as an infrastructure service. Fall 2000, PhD thesis.

[8] L. Chen, B. Vickers, J. Kim, T. Suda, E. Lesnansky. ATM and Satellite Distribution of Multimedia Educational Courseware. In *Proc. of the IEEE ICC*, 1996.

[9] Y. Chu, S. Rao, H. Zhang. A case for end system multicast. *Proc. of ACM SIGMETRICS*, June, 2000.

[10] Y. Chu, S. Rao, S. Seshan, H. Zhang. Enabling conferencing applications on the Internet using an overlay multicast architecture. *Proc. of ACM SIGCOMM*, 2001.

[11] H. Deshpande, M. Bawa, H. Garcia-Molina. Streaming live media over peer-to-peer network. Technical Report, Stanford University, August, 2001.

[12] P. Francis. YOID: Your Own Internet Distribution. http://www.aciri.org/yoid/. April, 2000.

[13] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, J. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. *Proc. of the 4th Symp. on Operating System Design and Implementation (OSDI)*, 2000.

[14] D. Kostic, A. Rodriguez, A. Vahdat. The Best of Both Worlds: Adaptivity in Two-Metric Overlay Networks. Duke University Technical Report, May, 2002.

[15] T. Nguyen and A. Zakhor. Distributed Video Streaming over the Internet. *Proc. of SPIE Conference on Multimedia Computing and Networking*, San Jose, CA, 2002.

[16] V. Padmanabhan, H. Wang, P. Chou, K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. *Proc. of 12th ACM NOSSDAV*, May, 2002.

[17] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: an application level multicast infrastructure. *Proc. of the 3d USENIX Symposium on Internet Technologies*, March, 2001.

[18] PlanetLab. http://www.planet-lab.org/

[19] S. Ratnasamy, M. Handley, R. Karp, S. Shenker. Topologically-aware construction and server selection. *Proc. of IEEE Infocom*, 2002.

[20] P. Rodriguez, A. Kirpal, and E. W. Biersack. Parallel-access for mirror sites in the Internet. *Proc. of IEEE Infocom*, 2000.

[21] P. Rodriguez, E. W. Biersack. Bringing the Web to the Network Edge: Large Caches and Satellite Distribution. ACM Special Issue. In *Journal on Special Topics in Mobile Networking and Applications (MONET)*, 2001 .

[22] A. Roy, S. Das. Optimizing QoS-Based Multicast Routing in Wireless Networks: A Multi-Objective Genetic Algorithmic Approach. *Proc. of Second International IFIP-TC6 Networking Conference (Networking'2002)*, LNCS vol. 2345, Springer-Verlag, Pisa, Italy, 2002.

[23] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. *Proc. of 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[24] vTrails. *http://www.vtrails.com/*