On Application-level Load Balancing in FastReplica

Jangwon Lee, Gustavo de Veciana

***Abstract*—**

In the paper, we consider the problem of distributing large-size content to a fixed set of nodes. In contrast with the most existing end-system solutions to this problem, FastReplica [1] does not attempt to build a 'good' overlay structure, but simply uses a fixed mesh overlay structure. This can significantly reduces the overheads incurred in probing, building and maintaining the overlay structure, otherwise. However, FastReplica is oblivious to heterogeneous and dynamic environments. To remedy this problem, we propose an *application-level load balancing* idea: putting more data on 'good' paths and less on 'bad' ones. Our goal is to study (1) how to make FastReplica adaptive to dynamic environments and (2) how much performance gain can be achieved by exploring the application-level load balancing idea in FastReplica.

Toward this end, we provide a theoretical analysis of a simplified model, which provides the insights serving as a basis to develop an implementation of this concept. Then, we present a performance evaluation on a wide-area testbed with a prototype implementation, showing that addition of application-level load balancing in FastReplica can achieve significant speedups by exploiting heterogeneous paths and dynamically adapting to bursty traffic.

keywords Communications/Networking and Information Technology, Network Protocols

I. Introduction

Content delivery networks (CDNs) are deployed to improve network, system and end-user performance by caching popular content on edge servers located close to clients. Since content is delivered from the closest edge server to a user, CDNs can save network bandwidth, overcome server overload problems and reduce delays to end clients. CDN edge servers were originally intended for static web content, e.g., web documents and images. Thus, if the requested content was not available or out-of-date, the local server would contact the original server, refresh its local copy and send it to the client. This *pull* type of operation works reasonably well for small to medium size web content, since the performance penalties for a cache miss, e.g., additional network traffic from the original server to the local server and higher delay to the client, are not significant. However, CDNs have recently been used to the deliver large files, e.g., digital movies, streaming media and software download packages. For large files, it is desirable to operate in a *push* model, i.e., replicating files at edge servers in advance of user's requests, since their distribution requires significant amounts of bandwidth. The download times may be high, e.g., 20 min media file encoded at 1 Mbit/s results in a 150 MBytes file or a high quality digital movie may be around 700 MBytes. Such push style file replication across distributed machines is also required for web mirror services.

In this paper we consider the problem of content distribution across geographically distributed nodes. Our focus is on distributing large files such as software packages or stored media files, and our objective is to minimize the overall replication time, i.e., minimizing the worst case download time to a set of receivers.[1]

Recently, an end-system approach [2], [3], [4], [5], [6], [7], has been emerged as a powerful method for delivering content while overcoming the deployment hurdles of IP multicast, e.g., reliability, router modification, and congestion control issues. In the end-system approach, hosts cooperate to construct an *overlay* structure consisting of unicast connections between end-systems. Since each overlay path is mapped onto a path in the underlying physical network, choosing and using good quality overlay paths when constructing overlay structures significantly impacts performance. Despite variations, most existing solutions based on the end-system approach, focus on finding and maintaining 'good' overlay structures (either tree or mesh) or reconfiguring them to optimize performance according to the application's requirements.

However, the above-mentioned flexibility in building overlay structures comes at a price. That is, building op-

J. Lee is with Qualcomm, Inc., and G. de Veciana is with Electrical and Computer Eng., The University of Texas at Austin. Corresponding Author: Jangwon Lee, 5775 Morehouse Drive, San Diego, CA 92121, Tel) 858-651-5185, Fax) 858-845-2651, E-mail: jangwonl@qualcomm.com

[1]Note that there exist many applications whose performance is determined by that of the worst case, e.g., especially in distributed environments where each node is responsible for a unique part of overall work based on the replicated data.

timized overlay structures requires path quality information among hosts. Since overlay paths may share common physical links, *sequential* probing to estimate available bandwidth on end-to-end paths (i.e., without the presence of other overlay path probing) may result in poor choice. If this is the case, *joint* probing over a large number of combinations should be performed, which may lead to huge overheads. After building an overlay structure, participants need to maintain it by exchanging control signals. To adapt to dynamic network situations, additional monitoring of alternative paths may be required. Furthermore, while restructuring happens, further overheads may be incurred due to lost packets or reconfiguration.

To expedite the delivery time in distributing large files in the context of a content delivery network, we proposed *FastReplica*, which is also an application level approach [1]. FastReplica uses two key ideas: (1) file splitting and (2) multiple concurrent connections. That is, the source divides the original file into m (the number of receivers) chunks, and concurrently transmits a different chunk to each receiver. Meanwhile each receiver relays its chunk to the remaining nodes so that each node ends up with all m chunks.[2]

In contrast with most existing end-system approaches, FastReplica does not attempt to build a "good" overlay structure, but simply uses all available paths, i.e., fixed m^2 overlay paths among the source and m receivers. The key feature associated with using a fixed overlay structure in this manner is that there is virtually no control overhead required with finding, maintaining and reconfiguring a good overlay structure. Furthermore, experiments on a wide-area testbed showed the potential of this approach to reduce the overall replication time [1].

However, FastReplica is oblivious to heterogeneity in the overlay paths, and simply puts an equal amount of data on each chunk. Heterogeneity in the overlay paths may arise due to the following factors. First, it is inherent in network resources. Infrastructure-based CDNs or web server replica networks are equipped with a dedicated set of machines and/or network links, which are engineered to provide a high level of performance. However, there are inherent heterogeneities among network resources, e.g., different capabilities of servers or available capacity on network links. Second, even with homogeneous network resource assumption, (e.g., nodes' capabilities and capacities of links between all end points are uniform), each chunk transfer may not achieve the same throughput since multiple overlay paths may be mapped onto the common physical links. Third, Internet traffic is variable. The available bandwidth on each path may vary with time possibly even during the file transfer.

The motivation of our work in this paper is to make FastReplica in the Small [1] adaptive to dynamic and heterogeneous environments. The key idea for it, is simple: introducing application-level load balancing in FastReplica framework [1], i.e., instead of putting the same amount of portion in each chunk, the source node puts more data on 'good' overlay paths and less data on 'bad' ones.

The goal of this paper is (1) to propose how this seemingly obvious and simple application-load balancing idea can be incorporated toward a practical solution, called Adaptive FastReplica (AFR), and (2) to study how much performance gain can be achieved by adding this application-load balancing idea into FastReplica.

Toward this end, we propose an analytical network model and study the optimal file partitioning rule which minimizes the overall replication time, i.e., the worst case download time. Furthermore, we show convergence for a proposed iterative algorithm towards an optimal partitioning. Despite the fact that this analysis makes some simplifications on the network and traffic models due to its tractability, it is of theoretical value and provides insights.

We evaluate the performance of our preliminary implementation on a real, but still somewhat controlled, Internet environment [8]. We find that application-level load balancing allows AFR to achieve significant speedups over FastReplica by enabling adaptivity to dynamic and heterogeneous Internet environments – these benefits are in turn significant when the network is highly dynamic.

The rest of the paper is organized as follows. In Section II, we introduce analytic models and then formulate and solve the optimization problem associated with minimizing the overall replication time. Section III proposes AFR mechanism and discusses its prototype implementation. This is followed by Section IV wherein we present our experimental results over a wide-area network environment. In Section V we discuss related work, and Section VI contains the conclusion and future work.

II. Analysis

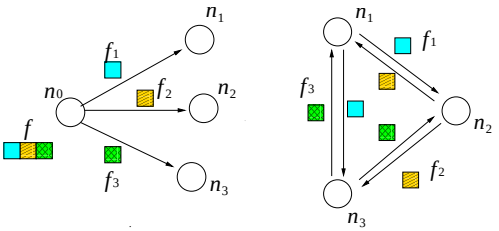
In this section, we present a network model and theoretical analysis to study application-level load balancing in FastReplica. The analytical results developed in this section will serve as a basis for AFR mechanism presented in Section III as well as the subsequent implementation.

A. Framework

Suppose a file f is available at a source node n_0 and is to be replicated across a set of receiver nodes, $R = \{n_i | i \in$

[2]This is FastReplica in the Small algorithm. To support a larger number of receivers in the group, FastReplica in the Small algorithm is applied iteratively using a hierarchical k-ary tree structure [1].

$N\}$, where $N = \{1,\ldots,m\}$ is the index set for receivers. Here, R is known to the source. Let f also denote the size of file in bytes. The file f is divided into m chunks, $f_1,\ldots,f_m$, such that $\sum_{i=1}^{m} f_i = f$ and $f_i \geq 0,\ i \in N$. In order to represent the portion of the original file that goes to each chunk, we define a *partition ratio vector*, $\mathbf{x} = (x_i = \frac{f_i}{f}, i \in N)$. Note that $0 \leq x_i \leq 1$ and $\sum_{i=1}^{m} x_i = 1$. Then, the file is replicated as follows.



(a) At the source. (b) At the receivers.

Fig. 1. An illustration of the replication framework.

• *At the source n_0:* Node n_0 opens m concurrent connections to each node in the replicating set, $n_1,\ldots,n_m$, and sends to each of these nodes, the following two items: (1) a list of replicating nodes, $\{n_i | i \in N\}$, and (2) its associated chunk of the file, i.e., chunk f_k is sent to node n_k. This procedure is shown in Figure 1(a) in the case where $m = 3$.

• *At each node n_k:* After receiving the list of replicating nodes, a node n_k opens $m-1$ concurrent connections to the remaining nodes, $R \setminus \{n_k\}$, and concurrently relays its chunk, f_k to each of them. This procedure is depicted in Figure 1(b). We assume relaying of data takes place on the fly, i.e., the node does not wait for the arrival of the entire chunk prior to relaying.

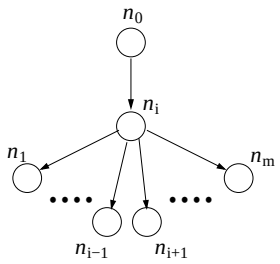


Fig. 2. Overlay tree of the i^{th} chunk.

Figure 2 shows the *overlay tree* traversed by data in chunk i. As can be seen, the tree includes *m overlay links (paths)*. An overlay link between two nodes may consist of multiple physical links and routers, i.e., it corresponds to a unicast path in the underlying physical network. Also, note that multiple overlay paths may share the same physical link. Since there are m relay nodes, there are m such trees. Thus, m^2 overlay links are used for the entire file transfer. It should be noted that this paper deals with FastReplica in the Small where m is small. If m is large, a hierarchical structure proposed in [1] could be used to achieve scalability. How to select such m value needs further future study.

We shall let $t_{ij}(\mathbf{x}),\ i,j \in N$, denote the *transfer time* of the i^{th} chunk from node n_0 to node n_j, when the partition vector $\mathbf{x}$ is used. We further define the *download time* to the j^{th} receiver, $d_j(\mathbf{x}) = \max_{i \in N} [t_{ij}(\mathbf{x})]$ and the *worst case transfer time* for the i^{th} chunk, $t_i(\mathbf{x}) = \max_{j \in N} [t_{ij}(\mathbf{x})]$. These times all depend on the partition ratio vector $\mathbf{x}$. For simplicity, we will occasionally suppress $\mathbf{x}$ in these notations.

B. Network Model

In our analytical network model, we assume that file transfer time is governed by the available bandwidth from a source to a destination node. This is reasonable when large files are being transferred [9]. For bandwidth availability, we assume a **tree** session model wherein chunk is transferred at the same rate along the entire overlay tree. This is not the case where each overlay path is realized by TCP connection. However, as can be seen in the sequel, the tree session model assumption provides us with not only tractability but also the key insights needed for an intuitive solution to the real-world problem.

We consider a network consisting of a set of links $\mathcal{L}$ with capacity $\mathbf{c} = (c_l, l \in \mathcal{L})$. We associate a *tree* session with each chunk. Let $\mathcal{S}$ denote the set of m tree sessions sharing the network. Bandwidth is allocated to each tree session, according to an appropriate criterion, e.g., max-min fair allocation [10], proportional fair allocation [11], max-throughput allocation [12], or that realized by coupled TCP sessions.

For $s_i \in A \subset \mathcal{S}$, we let $a^*_{s_i}(A)$ denote the bandwidth allocated to session s_i when the tree sessions in A *persist* on the network. Subsequently, we will call $A \subset \mathcal{S}$ the *active set*, i.e., the set of tree sessions which still have a backlog to send. For $s_i \in \mathcal{S}$, we let $a^*_{s_i}$ denote the bandwidth allocated to s_i when all m tree sessions are active in the system, i.e., $a^*_{s_i} = a^*_{s_i}(\mathcal{S})$.

Since the same bandwidth is allocated along all paths of the tree session for each chunk, the completion times under this model will satisfy $t_{ij} = t_i,\ \forall j \in N$, i.e., all receivers will get each chunk at the same time. Each tree session $s_i \in \mathcal{S}$ traverses a set of physical links $\mathcal{L}_{s_i}$ associated with the overlay tree for i^{th} chunk. Recall that there may be multiple-crossings of the same link by a given tree session, and such multiplicities can be easily accounted for. Since there is a one-to-one mapping between a chunk and a tree session, we will use these interchangeably in our notation, i.e., we will use a^*_i and $a^*_{s_i}$, or N and $\mathcal{S}$, interchangeably.

C. Optimal partitioning

If the partition ratios are the same for all chunks, i.e., $x_i = 1/m$, $i \in N$, the scheme in Section II corresponds to the FastReplica algorithm [1]. By controlling the partition ratio vector, the source can determine how much data is injected into each tree session, i.e., fx_i will be delivered through the i^{th} tree session. In this section, we formulate the following optimization problem assuming the network capacity is fixed (i.e., no other interfering traffic).

Problem 1: Suppose we are given a file f at a source node n_0 and a receiver set R. Under the tree session bandwidth allocation model, determine a partition ratio vector, $\mathbf{x} = (x_1, \ldots, x_m)$ which minimizes the overall replication time $r(\mathbf{x})$, given by

$$r(\mathbf{x}) = \max_{j \in N} [d_j(\mathbf{x})] = \max_{i,j \in N} [t_{ij}(\mathbf{x})] = \max_{i \in N} [t_i(\mathbf{x})].$$

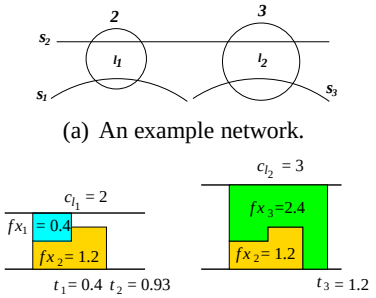
Depending on the partition ratio vector $\mathbf{x}$ and the network capacity, the bandwidths allocated for tree sessions may change dynamically over time. This is because if a session leaves the system (i.e., a chunk is successfully delivered), the network resources will be shared among the remaining sessions, possibly resulting in a new bandwidth allocation.

Let us consider an example to understand dynamics associated with Problem 1, i.e., how the transfer time of each chunk and the available bandwidth for each session are dynamically determined. We will suppose for simplicity that bandwidths among trees are allocated according to the max-min fair criterion[3] in our examples. Suppose the file size is $f = 4$ and the number of receivers is $m = 3$. Sessions s_1 and s_2 share a physical bottleneck link l_1 whose capacity is 2. Sessions s_2 and s_3 share a bottleneck link l_2 whose capacity is 3. The remaining links in the network are unconstrained and not shown in Figure 3(a).

Figure 3(b) shows each session's transfer time and assigned available bandwidth as time evolves when the partition ratio vector is $\mathbf{x} = (0.1, 0.3, 0.6)$. Times t_1, t_2, and t_3 in Figure 3(b) represent the transfer times of chunks 1, 2 and 3 respectively. Over time, we have the following active sets and bandwidth allocations:

$$\begin{aligned} 0 \le t \le t_1, \quad & A = \{s_1, s_2, s_3\}, a_1^*(A) = a_2^*(A) = 1 \\ & a_3^*(A) = 2 \\ t_1 < t \le t_2, \quad & A = \{s_2, s_3\}, a_2^*(A) = a_3^*(A) = 1.5 \\ t_2 < t \le t_3, \quad & A = \{s_3\}, a_3^*(A) = 3 \end{aligned}$$

[3]In max-min fair bandwidth allocation method, each session crossing a link should get as much as other such sessions sharing the link unless they are constrained elsewhere. Thus, it has the following characteristics: (1) each session has a bottleneck link, and (2) unconstrained sessions at a given link are given an equal share of the available capacity [10], [13].



(a) An example network.

(b) Max-min fair bandwidth allocation when $\mathbf{x} = (0.1, 0.3, 0.6)$ and $f = 4$.

Fig. 3. Illustration of dynamic bandwidth allocation.

For a given active set of sessions A, we let $g(A) = \sum_{i \in A} a_i^*(A)$ denote the *aggregate bandwidth*, i.e., the sum of bandwidths allocated to tree sessions in A. Then, the following provides an optimal solution to Problem 1.

Theorem 1: Suppose that A^* maximizes the aggregate bandwidth among all possible active sets in $\mathcal{S}$, i.e.,

$$A^* \in \operatorname*{argmax}_{A \subset \mathcal{S}} g(A) = \operatorname*{argmax}_{A \subset \mathcal{S}} \sum_{i \in A} a_i^*(A). \tag{1}$$

Then, x_i^* given by

$$x_i^* = \begin{cases} \frac{a_i^*(A^*)}{\sum_{k \in A^*} a_k^*(A^*)}, & i \in A^* \\ 0 & i \in \mathcal{S} \setminus A^* \end{cases}, \tag{2}$$

is an optimal solution to Problem 1.

Proof: Note that under the partition ratio x_i^*, all sessions in A^* complete at the same time, i.e., $\frac{fx_i^*}{a_i^*(A^*)} = \frac{f}{\sum_{k \in A^*} a_k^*(A^*)}$, for all $i \in A^*$. By definition, A^* offers the maximum instantaneous aggregate bandwidth rate to the receivers throughout the entire transfer. Thus, the proposed partition ratio vector must be optimal, though not necessarily unique. ■

From the perspective of the receivers, $g(A)$ is the amount of data per unit time they will get when all sessions in A are being used to transfer the file. A higher aggregate bandwidth will result in a lower overall replication time. Thus, Theorem 1 says that (1) we need to find which active set (tree session configuration) provides us with the maximal aggregate bandwidth, and (2) given such a set, say A^*, the partition ratio x_i^* satisfying Eq.(2) guarantees that the maximal aggregate bandwidth will be achieved throughout the file transfer. For example, for the network in Figure 3(a), we obtain $A^* = \{s_1, s_3\}$, $g(A^*) = 5$, $\mathbf{x}^* = (2/5, 0, 3/5)$, $r(\mathbf{x}^*) = \frac{f}{5}$. and for the network in Figure 4, we have $A^* = \{s_1, s_2, s_3\}$, $g(A^*) = 4$, $\mathbf{x}^* = (1/4, 1/4, 1/2)$, $r(\mathbf{x}^*) = \frac{f}{4}$.

Theorem 1 does not assert that the solution is unique, i.e., there may be multiple optimal solutions not satisfying the conditions in Theorem 1.

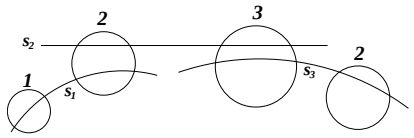


Fig. 4. Example networks.

III. AFR MECHANISM

In this section, we introduce key ideas underlying the AFR mechanism while gradually relaxing assumptions made in the analysis and considering a practical perspective. We then describe an AFR prototype implementation.

A. Full mesh structure

Theorem 1 suggests an optimal strategy based on the available bandwidth along overlay trees to minimize the overall file replication time. However, there are critical limitations to applying this result in practice. One of them is obtaining the active set A^* which maximizes the aggregate bandwidth. This is a complex combinatorial problem requiring either detailed knowledge of the network and tree structures, or possibly a brute force search over $2^m - 1$ possible solutions.

Instead of searching for an optimal active set, as with FastReplica, all tree sessions (m overlay trees) are used for the file transfer, i.e., $A = \mathcal{S}$, with the partition ratio vector suggested by Theorem 1:

$$x_i^+ = \frac{a_i^*}{\sum_{k\in\mathcal{S}} a_k^*}, \quad i \in \mathcal{S} \ . \tag{3}$$

Recall that $a_i^* = a_i^*(\mathcal{S})$ in Section II-B. Note that this solution may not be optimal. For example, for the cases in Figure 3(a) and Figure 4, this approach results in suboptimal and optimal solutions respectively.

Note that the partition ratio for the i^{th} chunk, x_i^+ is proportional to the bandwidth a_i^*. It is intuitive that for a 'bad' route (i.e., small a_i^*), a small chunk should be chosen, and for a 'good' route (i.e., large a_i^*), a large size chunk should be selected in order to reduce the overall replication time. Also recall that this partition ratio will make the transfer times for each chunk identical.

Under the model, the overall replication times for FastReplica (FR) and AFR would be[4]

$$r_{FR} \approx \frac{f}{m \min\limits_{i\in\mathcal{S}} a_i^*} \ , \quad r_{AFR} = \frac{f}{\sum_{i\in\mathcal{S}} a_i^*} \ . \tag{4}$$

Since $\sum_{i\in\mathcal{S}} a_i^* > m \min\limits_{i\in\mathcal{S}} a_i^*$, AFR will always beat FastReplica in terms of minimizing overall replication time. While the performance of FastReplica is limited by a single worst bottleneck link, the performance of AFR is limited by the sum of the m tree bottleneck links.

[4]Note that r_{FR} is an approximation in Eq.(4). This is because all chunks may not complete at the same time.

B. Block-level adaptation & In-band measurement

There are some more practical issues to be considered. First, in order to determine a partition ratio vector, a source needs prior knowledge of all the available bandwidths. A large amount of extra measurement traffic may need to be injected into the network to obtain accurate bandwidth estimation [14], [15]. Second, in the analysis in Section II-C, we assumed that the available network capacity for file transfers was fixed. However, this need not be the case in practice, i.e., the available bandwidth may change dynamically because of other traffic sharing the network. Even during a single file transfer, the available bandwidths for overlay paths may significantly change due to highly variable Internet traffic.

To adapt to such variations and reduce the overheads to obtain path quality information, the following block-level adaptation and in-band measurement approach is used. A large file is divided into multiple equal-sized *blocks*. Each block is again partitioned into m chunks based on average throughput information from past block transfers. Let $f(n)$ be the n^{th} block of file f. Throughout the rest of the paper, the number inside parenthesis represents an iteration step, e.g., $\mathbf{x}(n)$ is a partition ratio vector used for n^{th} block transfer and $t_i(n)$ is the worst transfer time for the i^{th} chunk for n^{th} block transfer.

The following describes the approach in more detail:

- For the first block transfer, the source uses an arbitrary partition ratio vector $x_i(1) > 0, i \in N$.
- At the $(n-1)^{th}$ iteration step ($n \geq 2$), each receiver n_j measures $a_{ij}(n-1) \equiv \frac{f(n-1)x_i(n-1)}{t_{ij}(n-1)}$, $i \in N$ defined as an *average throughput* achieved by the i^{th} chunk to the receiver n_j and sends it to the source – each receiver does this *for all* chunks in a block.
- At the n^{th} iteration step, the source updates the partition ratio vector $\mathbf{x}(n)$:

$$x_i(n) = \frac{a_i(n-1)}{\sum_{k=1}^m a_k(n-1)}, \ i \in N \tag{5}$$

where $a_i(n-1) = \min\limits_{j\in N} [a_{ij}(n-1)]$.

In the above approach, the theoretical available bandwidths are replaced with receiver estimates for average throughputs. See Eq. (5) vs. Eq (3). The intuition for this strategy is that a route is considered as 'good' if it achieved a high throughput and a route is 'bad' if it saw a low throughput.

The proposed in-band measurement has several practical advantages. First, the overhead to obtain path characteristics is low. It is easy to estimate average realized throughput rather than available bandwidth. Furthermore, since it uses an in-band measurement, this approach does not generate any extra traffic except feedback messages from receivers to the source. Second, the approach can provide more accurate path information as compared to *sequential* probing. Since overlay paths may share common physical links, sequential probing for estimation of available bandwidth between two nodes (i.e., without the presence of other overlay path probing) may result in poor estimation. In order to obtain more accurate path quality information, this requires joint probing with a number of combinations, which would incur a huge overhead. Since estimates for the average realized throughputs are obtained in the presence of other active overlay paths, they will provide fairly accurate path quality information.

Compared to IP multicast, a key advantage of end-system approach is the ability to adapt to changing network conditions. For example, one can reroute around congested or unstable areas of the Internet or reconfigure the overlay structures. However, in most existing end-system approaches, time-scale for those adaptive reconfiguration cannot be too small fine-grained due to large amount of control overhead during the reconfiguration phase. In contrast, the time scale of adaptivity in AFR roughly becomes the order of transferring a block. This enables AFR to achieve fine-grained adaptivity to dynamic Internet traffic.

The proposed approach does not incur a significant amount of control overhead: the total number of feedback messages generated in our solution for a single file transfer is $m^2\lceil\frac{f}{B}\rceil$ where B denotes the block size (each receiver generates $m\lceil\frac{f}{B}\rceil$ feedback messages). This overhead is further alleviated with TCP's piggyback mechanism [16], where a receiver does not have to send a separate TCP acknowledgment.

C. Convergence

With the proposed approach in Section III-B, initially, the source has no knowledge of the path characteristics. However, it learns path quality information from past block transfers, and uses these to update partition ratios. A natural question to ask is whether this procedure would converge, and if so, how quickly when the network capacity is static (i.e., there is no other interfering traffic). That is, irrespective of the initial chunk sizes, will the iterative algorithm converge to the partition ratio vector given in Eq. (3)?

1: INPUT: file f, list of receivers, block size B
2: send the list of receivers to R
3: initialize $x_i(1) = \frac{1}{m},\ i \in N$
4: **for** $n = 1$ to $\lceil\frac{f}{B}\rceil$ **do**
5: **if** there is a new latest throughput information $(a_i(j),\ i \in N,\ j < n)$ **then**
6: $x_i^{new}(n) \leftarrow \frac{a_i(j)}{\sum_{k=1}^{m} a_k(j)},\ i \in N$
7: $x_i(n) = (1-\alpha)x_i(n-1) + \alpha\, x_i^{new}(n)\ ,\ 0 \le \alpha \le 1$
8: **else**
9: $x_i(n) = x_i(n-1)$
10: **end if**
11: assign i^{th} chunk size based on $x_i(n)$ and concurrently send it to node n_i, $i \in N$ with size and block index information.
12: **end for**

Fig. 5. AFR algorithm for a source.

The difficulty in dealing with this question is that when the partition ratio is not optimal, different chunks will complete at different times, e.g., see Figure 3(b). Thus, one will obtain possibly biased estimates of the available bandwidth for tree sessions which see dynamically changing bandwidth allocations. Nevertheless, under simplifying assumptions one can show the convergence result which suggests such biases may not be problematic.

Theorem 2: Under the synchronous[5] iterative adaptation detailed in Section III-B with max-min fair bandwidth allocation across tree sessions, given any initial partition ratio vector $x_i(1) > 0,\ i \in N$, and a file of infinite length, the partition ratio $x_i(n)$ converges geometrically to x_i^+ given in Eq. (3).

Proof: See the Appendix.

Theorem 2 suggests that dynamic interactions among overlay paths sharing physical resources are not likely to lead to instabilities.

D. AFR prototype implementation

In this section, the prototype implementation of AFR is described. We have implemented AFR in multi-threaded C on the Linux system. As with most end-system approaches, each overlay path is realized by TCP connection. These connections are used to create sessions from the source to intermediate relays(receivers) which in turn create sessions to other receivers. The average throughput value associated with tree session i in our analysis, now corresponds to the worst case average throughput among m TCP connections relaying the i^{th} chunk.

[5] An extension to asynchronous updates subject to persistent updating in bounded time for each tree session should be straightforward [17].

```
1: receive the list of receivers
2: while file f not completely downloaded do
3:    concurrently keep reading data from n_0 and other
      receivers
4:    if data is coming from n_0 then
5:       forward them to other receivers
6:    end if
7:    if each chunk completely downloaded then
8:       send feedback message to n_0 containing its aver-
         age throughput value
9:    end if
10: end while
```

Fig. 6. AFR algorithm for receivers.

Figures 5 and 6 exhibit pseudo-codes describing the behaviors of the source and receiver respectively in AFR. The file is sent block by block. Each block is partitioned into m chunks depending on the current partition ratio vector. [6] In the AFR implementation, the source does not wait for the arrival of all feedback information from previous blocks prior to sending the next block – this should be contrasted with the approach discussed in Section III-B. To expedite the file transfer, the source keeps pushing blocks, and changes partitions only when the updated throughput information is available. In order to handle highly variable Internet traffic, the partition ratio vector was obtained by taking a weighted average between the previous partition ratio vector $x_i(n-1)$, and new estimate for the partition ratio vector $x_i^{new}(n)$ as shown on Line 7 in Figure 5. The impact of the moving average parameter α and block size is studied in Section IV-B.

IV. Experiments

We conducted experiments of the prototype of AFR implementation over the Internet testbed [8]. The conducted experiments are not intended to be representative of typical Internet performance, however, they do allow us to evaluate and validate the implementation and practical aspects of our approach. The goal for the experiments is to study (1) how the application-level load balancing can improve FastReplica, (2) what elements achieve such improvements, (3) the impact of setting parameters in AFR.

A. Experimental Setup and Metrics

Since the primary goal for the experiments is to evaluate the performance of AFR against FR, we attempted to run experiments in similar environments to those presented in [1]. We ran experiments by varying the source, receiver nodes' locations, and the number of participants, and report representative results involving the 9 hosts shown in Table I. Table II shows the five different content distribution configurations used to obtain the results below. For performance results of FastReplica with various configurations, refer to [1].

n_0	utexas.edu
n_1	columbia.edu
n_2	gatech.edu
n_3	umich.edu
n_4	ucla.edu
n_5	cam.ac.uk
n_6	caltech.edu
n_7	upenn.edu
n_8	utah.edu

TABLE I
PARTICIPATING NODES.

	source	receiver set
CONFIG 1	n_0	$n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8$
CONFIG 2	n_7	$n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_8$
CONFIG 3	n_0	n_1, n_2, n_3, n_4
CONFIG 4	n_7	n_0, n_5, n_6, n_8
CONFIG 5	n_0	n_1, n_7

TABLE II
CONFIGURATIONS.

The performance metrics we measured include: (1) the primary metric in the paper, *overall replication time*, $\max_{i \in N} d_i$, and (2) the *average replication time*, $\frac{1}{m}\sum_{k=1}^{m} d_k$. The measured times do not include any overheads that might be incurred due to the reintegration of chopped segments.

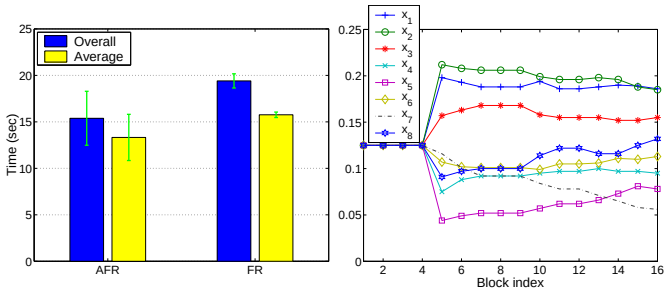
For ease of exposition, we define the percent *speedup* of scheme Y over scheme Z as $100(r_Z/r_Y - 1)\%$ where r_Z and r_Y are replication times of Z and Y schemes respectively. Thus, for example, 30% speedup of scheme Y over scheme Z means that scheme Y is 1.3 times faster than scheme Z. Unless explicitly mentioned, all results are averaged over 10 different runs and the vertical lines at data points represent 90% confidence intervals.

B. Performance results

Exploiting heterogeneous network paths. We examined performance results for AFR and FR under CONFIG 1. In this experiment, 8MB files are transferred from the source to receivers and we used a block size of 512KB with $\alpha = 0.1$ in AFR. (Later, we will discuss the impact of varying block size and α on the performance of AFR.) Figure 7(a) shows the overall and average replication times for AFR and FR. AFR outperforms FR: there are 26% and 18% speedups of AFR over FR for overall and average replication times respectively. We plot a particular realization of the partition ratio vectors for AFR as a function of the block index in Figure 7(b). Note that the summation of each element in partition ratio vectors is 1. In the beginning of a file transfer, the partition ratio vectors stay fixed at 0.125, i.e., source uniformly assigns traf-

[6] Since the ratios are not integer valued, the floor operation is used to assign the appropriate number of bytes to each chunk, that is, $f_i(n) = \lfloor Bx_i(n) \rfloor$. Then, the remaining bytes were distributed among chunks with the higher partition ratios in a round-robin manner.

fic load to each of chunk. Once feedback messages containing throughput information are available at the source, the source employs non-uniform partitions for next block transfer. Observe that partition ratios in AFR keep evolving attempting to converge to 'good' partitions reflecting inherent network path characteristics. In contrast, the partition ratio vectors for FR are fixed at 0.125 for the entire file transfer irrespective of quality of paths.



(a) Overall & Average time. (b) Partition ratio vectors for AFR.

Fig. 7. Performance results for AFR and FR under CONFIG 1.

The better performance of AFR results has several sources: first, AFR exploits inherent (static) heterogeneous network paths, and second, AFR adapts to dynamic traffic environments. Interestingly, we observed fairly tight confidence intervals on our measurements during our experiments, which indicates the network traffic loads (on PlanetLab) are fairly stable. To verify this, we also conducted the following experiments: we collected partition ratio vector values at the end of file transfers using AFR, and used them as the initial partition ratio values in FR scheme instead of equal partition ratio values. This FR-variant scheme can factor out the performance benefit obtained by adapting to bursty traffic loads, but holds performance benefit from being aware of inherent heterogeneity of network capacities. We observed that the replication times are slightly higher than but very close to those of AFR. We conclude that the performance benefits of AFR over FR mainly comes from being adapting to heterogeneous network paths rather than to more dynamic traffic loads. Note that using previous file transfer information to generate non-uniform initial partition ratio vectors as in this experiment can help AFR to quickly converge to correct values and eventually expedite file transfers.

In Section III, we show exponential rates of convergence, under a static scenario for a simplified model. Our goal however is to implement and use this scheme in practice, to see the adaptivity and convergence characteristics in a practical setting. In the practical setting, the convergence time can only be approximately inferred based on the dynamics of the partition ratio vector evolution, e.g., Figure 7(b). Specifically by observing the block index, where the partition ratio has converged, one can roughly infer convergence time. In Figure 7(b), the time taken from the beginning to the delivery of 6th block is 6.65 seconds, which can be considered as an approximate convergence time. However, note that the concept of the convergence rate in the theoretical model can not be exactly mapped into one in the real situation, since the experiment was performed in the actual network with possibly changing state due to dynamics in the network. Furthermore, in the theoretical model, when a source calculates the partition ratio vector at n^{th} block, the throughput information achieved at $(n-1)^{th}$ block is available. However, note that in the practical situation, this will not be the case since a source needs to keep sending blocks without waiting for the previous block's achieved throughput feedback.

Adapting to dynamic environments. Next we studied the potential benefit of AFR from being adaptive to dynamic traffic loads. For this, we ran the following two experiments on CONFIG 5 in Table II. For the first experiment, the source n_0 transfers an 8MB file to receivers, n_1 and n_7. For the second experiment, as with the first one, n_0 sends an 8MB file. However, 5 seconds after initiating the transfer, we deliberately generate interfering traffic by forcing n_7 to transmit a 32MB file to all the hosts in Table I except for n_0 and n_1. This interference traffic limit n_7's sending ability resulting in an increase in the replication time. Here, we used a block size of 128KB and $\alpha = 0.1$ for AFR.

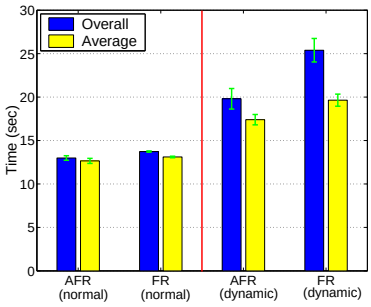
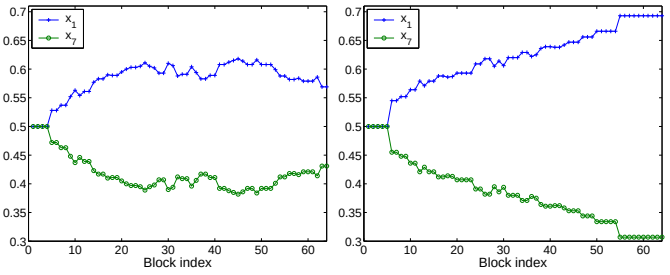


Fig. 8. Performance results for AFR and FR with/without dynamic environments.

Figure 8 exhibits the performance results for AFR and FR with and without traffic interference. In the first experiment without traffic interference, we see a 5% speedup gain of AFR over FR in the overall replication time. As with previous experiments, this gain comes from exploiting static heterogeneous network paths. However, in the second experiment with traffic interference a speedup of 28% was obtained. To study how these gains were achieved, we plot typical partition ratio vectors for the two experiments in Figure 9. Initially, both vectors fol-



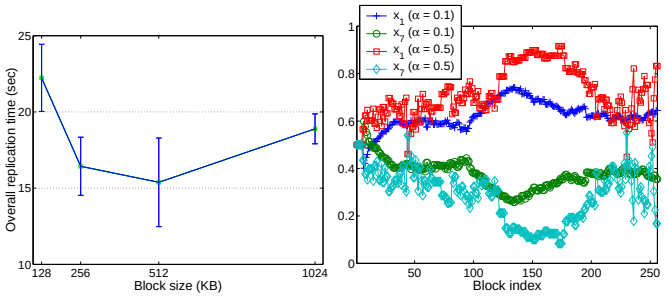
(a) Normal (b) Dynamic

Fig. 9. Partition ratio vectors with/without dynamic environments.

low similar patterns before around 20 blocks. However, while vectors of the normal case settle down to around constant value (0.6, 0.4) until the end of file transfer, those of dynamic case keep reducing the fraction of chunks associated with node n_7. This is because AFR can account for the limited sending ability of n_7 after 5 seconds and adapt to it. Note that there is no additional control overheads incurred to account for these dynamic environments. By simply placing different traffic loads at the source based on in-band measurement information, AFR effectively deals with varying Internet traffic potentially achieving significant performance benefits. We believe this feature of AFR can be more important in a highly dynamic network environment.

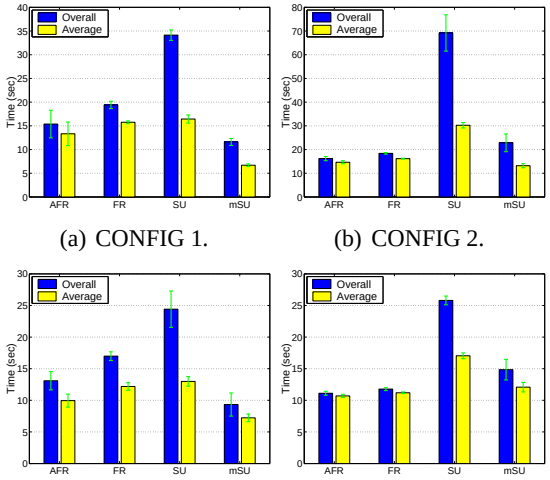
Varying parameters. We fix the moving average parameter α to 0.1 and change the block size from 128KB to 1024KB under CONFIG 1 to study the impact of different choices for the block size on the performance of AFR. Figure 10(a) shows the impact of varying block size on overall replication time when sending 8MB files. When blocks are two large, we lose opportunities to respond to dynamic changes in network bandwidth. At the extreme end where block size is equal to the file size, no adaptation is performed. On the other hand, there are large overheads incurred in processing feedback messages and reintegration when blocks are too small. Figure 10(a) illustrates this tradeoff.

To assess the impact of different α values on the performance of AFR, we varied α under CONFIG 1. However we did not see much change in the performance. This is because the network is fairly stable. Thus, we ran the following experiments to see how different α values behave in AFR: under CONFIG 5, the source sends 32MB size files, and 10 seconds later n_7 transfers 1MB files to $\{n_2, n_3, n_4, n_5, n_6, n_8\}$. We fixed block size to 128KB and tested $\alpha = 0.1$ and 0.5 values. Figure 10(b) shows partition ratio vectors for the experiments. We observe that both partition ratio vectors track down the change of network bandwidth during 1MB file transfer at n_7. That is, in the middle of file transfer, AFR puts a larger portion on the chunk going to n_1, but coming back to previous partition vectors after 1MB file transfers. (This again shows the ability of AFR to adapt to dynamic environments.) As expected, the value of α determines the degree of responsiveness. As with block size, there are tradeoffs in selecting α. A large α will quickly track changes in network state, but experience higher variability due to the measurement noise as shown in Figure 10(b). A small α is conservative and may not gain benefits from being adaptive to dynamic network changes.



(a) The impact of varying block size on the performance of AFR. (b) Partition ratio vectors for AFR with $\alpha = 0.1, 0.5$.

Fig. 10. The impact of varying parameters on the performance of AFR.



(a) CONFIG 1. (b) CONFIG 2.

(c) CONFIG 3. (d) CONFIG 4.

Fig. 11. Performance results under CONFIG 1-4.

The degree of heterogeneity. We conducted additional experiments under diverse configurations including two other schemes. *Sequential Unicast* (SU) measures the file transfer time from the source to each receiver *independently* via unicast (i.e., in the absence of other receivers), and then takes the worst case transfer time over all receivers assuming these times could be realized in parallel. SU is a hypothetical "optimistic" construction method for comparison purposes first proposed in [5]. *Multiple Sequential Unicast* (mSU) takes the same approach as SU

except that *m* (the number of receivers) concurrent unicast connections are used to realize the file transfer. That is, the file is equally partitioned into *m* chunks and those chunks are delivered via *m* parallel connections from the source to the receiver. We use this hypothetical scheme to provide lower bounds on performance.

Figures 11(a)-(d) show the performance results under CONFIG 1-4. We find that AFR consistently outperforms FR over all configurations. We observe that the degree of performance gains are proportional to the amount of heterogeneity in the overlay paths. The more heterogeneous overlay paths are, the more performance gains can be obtained using AFR. AFR and FR outperforms SU under all configurations. The key features of AFR and FR over SU are the use of (1) concurrent multiple connections and (2) diverse paths. The performance benefits resulting from these two features are extensively discussed in [1]. Perhaps surprisingly in our experiments we found the performance of AFR and FR was better than that of mSU under CONFIG 2 and 4. Since mSU employs multiple concurrent connections as with AFR and FR, we can conclude that this performance benefit comes from the path diversities offered by AFR and FR.

V. Related Work

A large amount of research has recently been pursued towards supporting multicast functionality at the application layer. In this section, we review work sharing similar goals as ours, i.e., specifically targeting bandwidth-intensive applications.

Overcast [4] organizes nodes into a source-rooted multicast tree with the goal of optimizing bandwidth usage across the tree. The work in [18] proposes an optimal tree construction algorithm based on the assumption that congestion only arises on access links and there are no lossy links. These two approaches are based on building a *single* high bandwidth overlay tree using bandwidth estimation measurements. Accurate bandwidth estimation requires extensive probing [14], [15]. Furthermore, since only a single multicast tree is used, the selection of the tree significantly impacts performance.

Recently, several approaches employing file splitting and multiple peer connections over multiple tree or mesh structures have been proposed [19], [20], [21], [22], [1]. Splitstream [20] splits the content into *k* stripes, sending them over a forest of interior-node-disjoint multicast trees. The focus in this work is on constructing multicast trees such that each intermediate node belongs to at most one tree while distributing forward load subject to bandwidth constraints. In Splitstream, tree construction and maintenance are done in a distributed manner using Scribe [23]. In Bullet [19], nodes initially construct a tree structure, but then use additional parallel downloads from other peers. A sender disseminates different objects to different nodes. Thus, RanSub [24] is used to locate peers that have disjoint content. Even after locating peers with disjoint content, Bullet requires a reconciliation phase to avoid receiving redundant data. This reconciliation is done using the approach proposed in [25]. BitTorrent [21] and Slurpie [22] find peers using dedicated servers, called *trackers* and *topology servers* respectively, whose role is to direct a peer to a random subset of other peers which already have portions of the file. In both schemes, a random mesh is formed among peers to download the file. Trackers in BitTorrent may have a scalability limit, as they continuously update the distribution status of the file. The Slurpie approach adapts to varying bandwidth conditions, and scales its number of peers in the subset based on estimating bandwidth and group size.

In contrast, the target environment for FastReplica [1] and AFR is for push-driven CDNs or Web cache systems since a sender initiates file transfer and knows the receivers a priori. FastReplica [1] and AFR can be viewed as an extreme form of exploiting multiple parallel connections - a full mesh structure is used. It is not unusual for less resource intensive techniques to evolve into more resource intensive ones as processing and storage become inexpensive, and if they provide additional flexibility. The transition from IP multicast to an application-level multicast and from a single tree multicast to multiple multicast trees or mesh structures follows this trend. Furthermore, since CDNs or Web cache systems are usually equipped with dedicated high performance network resources, their concerns are at how to fully utilize their resources.

Note that most schemes employing file splitting use a variety of encoding schemes, e.g., erasure codes [26], [27] or Multiple Description Coding (MDC) [28] to efficiently disseminate data or recover from losses. These types of encoding schemes can also be used in AFR.

VI. Conclusions and Future work

In this paper, we explored the application-level load balancing idea over a fixed overlay structure in FastReplica framework: exploiting 'good' paths by putting more data on them. The salient feature of FastReplica, which differentiates it from existing approaches, is that it requires virtually no control overhead to construct overlay structure. Via theoretic analysis and experiments, our work showed that addition of application-level load balancing in FastReplica can achieve significant speedup by enabling fine-grained adaptivity to dynamically varying

Internet traffic without the need for reconfiguring the overlay structure.

Here, we conclude our paper with enumerating some interesting future topics that remain for FastReplica framework. The first one is for FastReplica in the "Large" [1]. In [1], we suggested that if there are a large number of receivers, a hierarchical *k*-ary tree might be constructed. Note that in the FastReplica in the "Small", we deliberately chose to ignore topological properties among members to eliminate the overhead for constructing an overlay structure. However, topology closeness within a replicating group in FastReplica in the Large, is important. How one should select *k* and how one should cluster the nodes in the replication groups, are the important research topics to support an efficient FastReplica-style file distribution in a large-scale environment.

Second one is to do more extensive experimentations for performance results comparing with various file distribution systems mentioned in Related Work section. Even though the target environment is somewhat different, such experiments would provide a quantitative comparison and pros / cons of each solution. Moreover, extensive performance experiments could lead optimiztion on the selection of parameters (α and block size) and provide more concrete recommendation based on different input parameters and environments.

Lastly, extending the proposed application-level load balancing technique in FastReplica to peer-to-peer type applications, is an interesting future topic.

REFERENCES

[1] L. Cherkasova and J. Lee, "FastReplica: Efficient large file distribution within content delivery network," in *4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[2] P. Francis, "Yoid: Extending the Internet multicast architecture," in *Tech. reports, ACIRI, http://www.aciri.org/yoid*, 2000.

[3] Y. Chawathe, *Scattercast: An architecture for Internet broadcast distribution as an infrastructure service*, Ph.D. thesis, University of California, Berkeley, 2000.

[4] J. Jannotti, D. Gifford, K. Johnson, F. Kasshoek, and J. O'Toole, "Overcast: Reliable multicasting with an overlay network," in *USENIX OSDI*, 2000.

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

[6] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: an application level multicast infrastructure," in *3rd USENIX Symposium on Internet Technologies and Systems (USITS)*, 2001.

[7] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM*, 2001.

[8] L. Peterson, T. Anderson, D. Culler, and T. Roscoe, "A blueprint for introducing disruptive technology into the Internet," in *Proceedings of the First ACM Workshop on Hot Topics in Networks*, 2002.

[9] M. Sharma and J. W. Byers, "How well does file size predict wide-area transfer time?," in *Global Internet*, 2002.

[10] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, 1992.

[11] F. P. Kelly, A. K. Maullo, and D. K. H. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," in *Journal of Operational Research Society*, 1998.

[12] J. Roberts and L. Massoulié, "Bandwidth sharing and admission control for elastic traffic," in *ITC Specialist Seminar*, 1998.

[13] T. Lee, *Traffic management & design of multiservice networks: the Internet & ATM networks*, Ph.D. thesis, The University of Texas at Austin, 1999.

[14] K. Lai and M. Baker, "Measuring link bandwidths using a deterministic model of packet delay," in *Proc. ACM SIGCOMM*, 2000.

[15] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with tcp throughput," in *Proc. ACM SIGCOMM*, 2002.

[16] D. E. Comer, *Internetworking with TCP/IP*, Prentice Hall.

[17] D. Bertsekas and J. Tsitsiklis, *Paralle and Distributed Computation:Numerical Methods*, Prentice Hall, 1997.

[18] M. S. Kim, S. S. Lam, and D. Lee, "Optimal distribution tree for internet streaming media," in *Proc. IEEE International Conference on Distributed Computing Systems*, 2003.

[19] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *Proc. of ACM Symposium on Operating Systems Principles*, 2003.

[20] M. Gastro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth content distribution in cooperative environments," in *Proc. of ACM Symposium on Operating Systems Principles*, 2003.

[21] "Bittorrent," `http://www.bitconjurer.org/BitTorrent`.

[22] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A cooperative bulk data transfer protocol," in *IEEE Infocom*, 2004.

[23] A. Rowstron, , A. Kermarrec, M. Gastro, and P. Druschel, "SCRIBE: The design of a large-scale event notification infrastructure," in *Networked Group Communication*, 2001.

[24] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat, "Using random subsets to build scalable network services," in *4th USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[25] J. Byers, J. Considine, M. Mitzenmacher, and S. Rost, "Informed content delivery across adaptive overlay," in *Proc. ACM SIGCOMM*, 2002.

[26] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *ACM SIGCOMM*, 1998.

[27] M. Luby, "LT codes," in *The 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002.

[28] V. K. Goyal, "Multiple description coding: Compression meets the network," in *IEEE Signal Processing Magazine*, 2001.

[29] D. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation:Numerical methods*, Prentice Hall, 1989.

APPENDIX

Notations: Here we formally define the hierarchy bottlenecks related to max-min fairness that will be useful in the sequel. We define the *fair share* $y_l^1 = c_l/m_l^1$ at a link $l \in \mathcal{L}$ as a fair partition of capacity at the link in the 1^{st} level of the hierarchy, where $m_l^1 = |\mathcal{S}_l|$ is the number of

sessions through l. Then, the set of 1^{st} level bottleneck links and sessions are defined as follows:

$$\begin{aligned}\mathcal{L}^{(1)} &= \{l \in \mathcal{L} | \forall s \in \mathcal{S}_l,\ a_s^* = b^{(1)} = \min_{k \in \mathcal{L}} y_k^1\},\\ \mathcal{S}^{(1)} &= \{s \in \mathcal{S} | s \in \mathcal{S}_l \text{ and } l \in \mathcal{L}^{(1)}\}.\end{aligned}$$

where a_s^* is the bandwidth assigned for the session s. Thus $\mathcal{L}^{(1)}$ is the set of 1^{st} level bottleneck links such that the sessions in $\mathcal{S}^{(1)}$ traversing these links are allocated the minimum bandwidth ('fair share') in the network, denoted by $b^{(1)}$. These two sets make up the 1^{st} level of the bottleneck hierarchy.

The next level of the hierarchy is obtained by applying the same procedure to a reduced network. The reduced network is obtained by removing the sessions in $\mathcal{S}^{(1)}$. The capacity at each link in $\mathcal{L} \setminus \mathcal{L}^{(1)}$ traversed by sessions in $\mathcal{S}^{(1)}$ is reduced by the bandwidth allocated to these sessions. The bottleneck links $\mathcal{L}^{(1)}$ are also removed from the network. Thus $\mathcal{L}^{(2)}$ and $\mathcal{S}^{(2)}$ are obtained based on a network with fewer links and sessions and adjusted capacities. The set of 2^{nd} level bottleneck links, sessions, and bottleneck rate $b^{(2)}$ are defined as before. This process continues until no sessions remain.

Let $\mathcal{U}^{(h)} = \cup_{k=1}^{h} \mathcal{L}^{(k)}$ and $\mathcal{V}^{(h)} = \cup_{k=1}^{h} \mathcal{S}^{(k)}$. These are defined as the cumulative sets of bottleneck links and sessions, respectively, i.e., for levels 1 to h of the hierarchy. The fair share y_l^h $(h \geq 2)$ of link l in $l \in \mathcal{L} \setminus \mathcal{U}^{(h-1)}$ is defined as the fair share of the available capacity at the link in the h^{th} level of the hierarchy:

$$y_l^h = \frac{c_l - \beta_l^{h*}}{m_l^h} \tag{6}$$

where $\beta_l^{h*} = \sum_{s \in \mathcal{S} \cap \mathcal{V}^{(h-1)}} a_s^*$ is the total flow of sessions through l which are constrained by bottleneck links in $\mathcal{U}^{(h-1)}$, and $m_l^h = |\mathcal{S}_l \setminus \mathcal{V}^{(h-1)}|$, where $m_l^h > 0$, is the number of sessions through l which are unconstrained by the links in $\mathcal{U}^{(h-1)}$. Based on the fair share, the set of h^{th} level $(h > 2)$ bottleneck links and sessions can be defined as:

$$\begin{aligned}\mathcal{L}^{(h)} &= \{l \in \mathcal{L} \setminus \mathcal{U}^{(h-1)} |\\ &\forall s \in \mathcal{S}_l,\ a_s^* = b^{(h)} = \min_{k \in \mathcal{L} \setminus \mathcal{U}^{(h-1)}} y_k^h\},\\ \mathcal{S}^{(h)} &= \{s \in \mathcal{S} \setminus \mathcal{V}^{(h-1)} | s \in \mathcal{S}_l \text{ and } l \in \mathcal{L}^{(h)}\}.\end{aligned} \tag{7}$$

Here $\mathcal{L}^{(h)}$ is the set of h^{th} level bottleneck links such that the sessions in $\mathcal{S}^{(h)}$ are allocated the minimum fair share in the reduced network. We repeat this procedure until we exhaust all the links and sessions resulting in a hierarchy of bottleneck links and corresponding sessions $\mathcal{L}^{(1)}, \ldots, \mathcal{L}^{(q)}$ and $\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(q)}$, which is uniquely defined by (6) and (7), where q is the number of levels in the hierarchy.

In the sequel, when we refer to the hierarchy of bottlenecks we mean the *initial* hierarchy structure when all sessions are active, i.e., $A = \mathcal{S}$. Consider an h^{th} level link $l \in \mathcal{L}^{(h)}$ of this bottleneck hierarchy. Note that the sessions sharing link l can be partitioned into those in levels 1 to h, i.e., $\mathcal{S}_l = \cup_{j=1}^{h} [\mathcal{S}_l \cap \mathcal{S}^{(j)}]$. For sessions in the j^{th} level of the hierarchy, suppose we order them by their finishing times (i.e., when they depart from the system) on the n^{th} file transfer. We let $s_i^j(n)$ denote the i^{th} session to leave the system among j^{th} level sessions at the n^{th} step.[7] Thus we have that

$$t_{s_1^j}(n) \leq t_{s_2^j}(n) \leq \ldots \leq t_{s_k^j}(n),$$

where $k = m_l^j$. Note that at each iteration step, the order in which sessions complete may change. For the session $s_i^h(n)$, we let $p^j(s_i^h(n))$ be the number of sessions at the j^{th} level whose departure times are equal or less than that of session $s_i^h(n)$. Then, note that at time $t_{s_i^h}(n)$, the number of remaining j^{th} level flows in the system is $m_l^j - p^j(s_i^h(n))$ since m_l^j is the total number of j^{th} level sessions in link l. Figure 12 illustrates the notation we have defined.

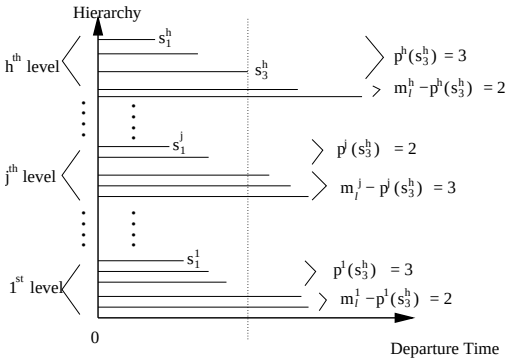


Fig. 12. Sessions in the h^{th} level link.

Proof of Theorem 2. We will prove Theorem 2 by induction on the *initial* bottleneck hierarchy which is constructed when all sessions are in the system. Without loss of generality, we let the file size be 1, i.e., $f(n) = 1$, $n \geq 1$.

Step 1: Consider a 1^{st} level bottleneck link $l \in \mathcal{L}^{(1)}$. Note that for any session s in $\mathcal{S}_l$ and at any iteration step n,

$$b^{(1)} \leq a_s(n). \tag{8}$$

Indeed the bandwidth rates for all first level sessions are non-decreasing over time. That is, once sessions start to

[7] For simplicity, we may suppress an iteration step index in notations, e.g., $t_{s_i^j(n)}(n) = t_{s_i^j}(n)$.

leave the link l, the additional bandwidth available to the remaining sessions can only result in an increase in the bandwidth allocated to a first level bottleneck session. For s_i^1, the i^{th} session to leave the system among the first level sessions on link l, we have a lower bound on its transfer time as follows:

$$\underline{t}_{s_i^1}(n) = \frac{\sum_{k=1}^{i} x_{s_k^1}(n)}{c_l - (m_l^1 - i)b^{(1)}} \leq t_{s_i^1}(n). \tag{9}$$

The numerator is the amount of work to be done from the first to the i^{th} session. The denominator is the largest amount of bandwidth available for transferring this data. This is because during $t_{s_i^1}(n)$ there are at least $(m_l^1 - i)$ flows each with a bandwidth allocation of at least $b^{(1)}$. Thus we have that

$$\begin{aligned} b^{(1)} \leq a_{s_i^1}(n) &= \frac{x_{s_i^1}(n)}{t_{s_i^1}(n)} \leq \frac{x_{s_i^1}(n)}{\underline{t}_{s_i^1}(n)} \\ &= \frac{x_{s_i^1}(n)}{\sum_{k=1}^{i} x_{s_k^1}(n)}[c_l - (m_l^1 - 1)b^{(1)}] \qquad (10) \\ &= \frac{a_{s_i^1}(n-1)}{\sum_{k=1}^{i} a_{s_k^1}(n-1)}[c_l - (m_l^1 - 1)b^{(1)}] \qquad (11) \\ &\leq \frac{a_{s_i^1}(n-1)}{(i-1)b^{(1)} + a_{s_i^1}(n-1)}[c_l - (m_l^1 - 1)b^{(1)}]. \qquad (12) \end{aligned}$$

We obtain Eq. (11) from Eq. (10) by using Eq. (5) and we have Eq. (12) from Eq. (11) by Eq. (8). Thus,

$$\begin{aligned} |a_{s_i^1}(n) - b^{(1)}| &\leq \left| \frac{a_{s_i^1}(n-1)}{(i-1)b^{(1)} + a_{s_i^1}(n-1)} \right. \\ &\qquad \left. [c_l - (m_l^1 - 1)b^{(1)}] - b^{(1)} \right| \qquad (13) \\ &= \left| \frac{b^{(1)}(i-1)(a_{s_i^1}(n-1) - b^{(1)})}{(i-1)b^{(1)} + a_{s_i^1}(n-1)} \right| \qquad (14) \\ &\leq \frac{i-1}{i} |a_{s_i^1}(n-1) - b^{(1)}|. \qquad (15) \end{aligned}$$

Eq. (15) is obtained from Eq. (14) by using $a_{s_i^1}(n-1) \geq b^{(1)}$. Let $\xi_l = \frac{m_l^1 - 1}{m_l^1}$. Then, $\forall\, s \in \mathcal{S}_l$ we have that

$$|a_s(n) - b^{(1)}| \leq \xi_l |a_s(n-1) - b^{(1)}|. \tag{16}$$

Since $0 < \xi_l < 1$, $a_s(n)$ converges geometrically to $b^{(1)} = a_s^*$.

Step 2: Suppose that for all $s \in \mathcal{V}^{(h-1)}$, $a_s(n)$ converges geometrically to a_s^*. Consider an h^{th} level link $l \in \mathcal{L}^{(h)}$ and a $s_i^h \in \mathcal{S}_l \cap \mathcal{S}^{(h)}$, which is the i^{th} session to leave the system among h^{th} level sessions in link l at iteration step n. We will show that $a_{s_i^h}(n),\ i \in \{1, \ldots, m_l^h\}$ will converge to $b^{(h)}$. This can be done by finding a lower bound and an upper bound and showing the bounds converge to $b^{(h)}$.

Convergence of the lower bound: Let $t_l^{\min}(n) = \min_{s \in \mathcal{S}_l \cap \mathcal{V}^{(h-1)}} [t_s(n)]$ and $t_l^{\max}(n) = \max_{s \in \mathcal{S}_l \cap \mathcal{V}^{(h-1)}} [t_s(n)]$. Thus, $t_l^{\min}(n)$ and $t_l^{\max}(n)$ are the earliest and the last departure time among sessions from levels 1 to $(h-1)$ at link l respectively. Now, define $\varepsilon_l^h(n)$ as the difference between these two times, i.e, $\varepsilon_l^h(n) = t_l^{\max}(n) - t_l^{\min}(n)$. Consider the time-varying bandwidth allocation for an h^{th} level session depicted in Figure 13.

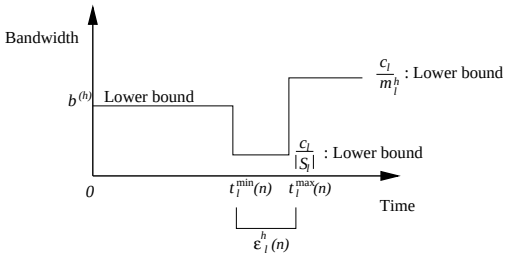


Fig. 13. Bandwidth allocation for an h^{th} level session.

- From the beginning of the transfer to $t_l^{\min}(n)$, no session in $\mathcal{S}_l \cap \mathcal{V}^{(h-1)}$ leaves link l. Thus, at least $b^{(h)}$ is allocated to an h^{th} level session during this period.
- After $t_l^{\max}(n)$, there are only h^{th} level sessions remaining, which traverse link l. A lower bound on bandwidth allocation for such sessions is c_l/m_l^h since m_l^h is the entire number of h^{th} level sessions at the beginning. Note that the lower bound is larger than $b^{(h)}$ by Eq. (6).
- It is possible for the h^{th} level session to be less than $b^{(h)}$ only during the period from $t_l^{\min}(n)$ to $t_l^{\max}(n)$. This may happen if sessions lower than h^{th} level change their bottleneck links to link l during this period. For example, consider Figure 3(b). Initially, the bottleneck hierarchy is preserved before session s_1 leaves the system. However, once s_1 departs, session s_2 changes its bottleneck link from l_1 to l_2 and equally sharing the capacity of l_2 with s_3. This results in reduction of s_3's bandwidth from 2 to 1.5. During this period of length $\varepsilon^h(n)$, we can have a lower bound on bandwidth for an h^{th} level session by $\frac{c_l}{|S_l|}$. This lower bound is the first share of the link l, i.e., y_l^1. Based on the above observations, we have the following lower bound for the average throughput of the any h^{th} level session:

$$\begin{aligned} \underline{a}_l^h(n) &= \frac{b^{(h)} t_l^{\min}(n) + \frac{c_l}{|S_l|}\varepsilon_l^h(n)}{t_l^{\max}(n)} \\ &= b^{(h)} - \frac{(b^{(h)} - \frac{c_l}{|S_l|})\varepsilon_l^h(n)}{t_l^{\max}(n)} \equiv b^{(h)} - \delta_l^h(n) \qquad (17) \\ &\leq a_s(n),\ s \in \mathcal{S}_l \cap \mathcal{S}^{(h)}. \end{aligned}$$

The above lower bound is the average throughput at time $t_l^{\max}$ in Figure 13, and it is a worst case scenario including a maximal amount time $\varepsilon_l^h(n)$ at the lower rate $\frac{c_l}{|s_l|}$. By our induction hypothesis, for $s \in \mathcal{V}^{(h-1)}$, $a_s(n)$ converges geometrically to a_s^*, so $\varepsilon_l^h(n)$ also converges to 0. Also note that as $\varepsilon_l^h(n)$ goes to 0, $\delta_l^h(n)$ also goes to 0 and eventually the lower bound $\underline{a}_s^h(n)$ converges geometrically to $b^{(h)}$.

Convergence of the upper bound: Next, consider the following lower bound for the finishing time of session s_i^h:

$$\underline{t}_{s_i^h}(n) = \frac{\sum_{j=1}^{h}\sum_{k=1}^{p^j(s_i^h)} x_{s_k^j}(n)}{c_l - \sum_{j=1}^{h}(m_l^j - p^j(s_i^h))\underline{a}_l^j(n-1)} \le t_{s_i^h}(n). \quad (18)$$

As with Eq. (9) in Step 1, the numerator is the amount of work to be done prior to s_i^h's departure and the denominator is an upper bound on the available bandwidth. In a similar manner to Step 1, we have that

$$\underline{a}_l^h(n) \le a_{s_i^h}(n) \le \frac{a_{s_i^h}(n-1)[c_l - \sum_{j=1}^{h}(m_l^j - p^j(s_i^h))\underline{a}_l^j(n-1)]}{\sum_{j=1}^{h-1} p^j(s_i^h)\underline{a}_l^j(n-1) + (p^h(s_i^h)-1)\underline{a}_l^h(n-1) + a_{s_i^h}(n-1)}. \quad (19)$$

By replacing Eq. (17) into Eq. (19) and using the fact that $c_l = \sum_{j=1}^{h} m_l^j b^{(j)}$, we have that

$$a_{s_i^h}(n) \le \frac{a_{s_i^h}(n-1)[\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} + P(n-1)]}{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} - Q(n-1) + a_{s_i^h}(n-1)},$$

where $P(n) = \sum_{j=1}^{h}(m_l^j - p^j(s_i^h))\delta_l^j(n)$ and $Q(n) = \sum_{j=1}^{h-1} p^j(s_i^h)\delta_l^j(n) - \delta_l^h(n)$. Now letting $R(n) = \max[P(n), Q(n)]$, we have an upper bound for an average throughput:

$$a_{s_i^h}(n) \le \frac{a_{s_i^h}(n-1)[\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - R(n-1)]}{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} + R(n-1) + a_{s_i^h}(n-1)} \equiv \bar{a}_{s_i^h}(n) \equiv T(R(n-1), a_{s_i^h}(n-1)) \quad (20)$$

Now we will show that $\bar{a}_{s_i^h}(n)$ converges to $b^{(h)}$ geometrically. Consider $T(\cdot, a_{s_i^h})$. We have that

$$\max_{R\ge 0}\left|\frac{\partial}{\partial R}T(R, a_{s_i^h})\right| \equiv A(a_{s_i^h}) = \left|\frac{-2a_{s_i^h}\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} + a_{s_i^h}b^{(h)} - (a_{s_i^h})^2}{[\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} + R(n-1) + a_{s_i^h}(n-1)]^2}\right|_{R=0}$$

For any $\varepsilon > 0$, by our lower bound, there is an n such that $a_{s_i^h}(n) \ge \underline{a}_{s_i^h}(n) \ge b^{(h)} - \varepsilon$. Thus, letting the Lipschitz constant $\bar{K} = A(b^{(h)} - \varepsilon)$, we have that

$$|T(R(n), a_{s_i^h}(n)) - T(0, a_{s_i^h}(n))| \le \bar{K}|R(n) - 0| \quad (21)$$

Note that $R(n)$ is a linear combination of $\delta_l^j(n)$, so $R(n)$ will converge geometrically to 0. Furthermore, $T(R,\cdot)$ is a pseudo-contraction [29], which converges to $T(0, b^{(h)}) = b^{(h)}$, i.e.,

$$\begin{aligned} &|T(0, a_{s_i^h}(n)) - T(0, b^{(h)})| \le \\ &\left|\frac{a_{s_i^h}(n)[\sum_{j=1}^{h} p^j(s_i^h)b^{(h)}]}{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} + a_{s_i^h}(n)} - b^{(h)}\right| \\ &\le \left|\frac{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)}}{\sum_{j=1}^{h} p^j(s_i^h)b^{(j)} - b^{(h)} + a_{s_i^h}(n)}\right| |a_{s_i^h}(n) - b^{(h)}| \\ &\le \xi|a_{s_i^h}(n) - b^{(h)}|, \ 0 < \xi < 1 \end{aligned} \quad (22)$$

So,

$$\begin{aligned} &|a_{s_i^h}(n+1) - b^{(h)}| = |T(R(n), a_{s_i^h}(n)) - T(0, b^{(h)})| \\ &\le |T(R(n), a_{s_i^h}(n)) - T(0, a_{s_i^h}(n))| + |T(0, a_{s_i^h}(n)) - T(0, b^{(h)})| \\ &\le \bar{K}|R(n) - 0| + \xi|a_{s_i^h}(n) - b^{(h)}|, \quad 0 < \xi < 1. \end{aligned}$$

Thus, $a_{s_i^h}(n)$ converges geometrically to $b^{(h)}$. This completes the proof.