Seamless Self-Assembly of Files in Cache Networks at Minimal Storage Cost

Andreas Constantinides, *Student Member, IEEE,* and E.G. Coffman, Jr., *Fellow, IEEE*

***Abstract*— A file F is sited at the root of a tree network of nodes, each able to cache part or all of F. Each node may also have a client wanting to download F. Links between neighbors in the tree have given transmission delays associated with them, but there is no transmission delay from a client to the node where it is located. The paradigm of seamless transmission requires that requesting nodes start receiving F immediately, receive it continuously until F is fully assembled, and be implemented by means of a protocol which is unaware of network structure beyond links to immediate neighbors. Because of similar processes known as *self-assembly* in biology and molecular science, it is natural to adopt the term *seamless self-assembly* for present purposes.**

Nodes where clients are located are specified as part of the problem instance. We formulate a discrete-time model and show how to assign segments of F to the node caches in such a way that seamless self-assembly is realized and, simultaneously, the total cache size achieves a lower bound determined solely by the network link delays. Extensions of the problem in several dimensions are discussed as directions for further research.

I. Introduction

A new paradigm in information transmission was introduced in [1] for networks of cooperative cache servers[1]. Each file in the network has an origin server which stores the entire file, and all nodes of the network are potential *client nodes* for that file, i.e., nodes having clients wanting that file. A file segment of any size can be cached at any of the nodes. The downloading of a file to a client is *seamless* in that it incurs no delay, i.e., it begins when the request is made and continues without interruption until it is complete. Thus, seamless transmission can be viewed as a way to implement a maximum quality of service (QoS) guarantee to all clients of a network. We study this problem for a single file F, under a distributed, *self-assembly* constraint in which there is no global control and nodes are unaware of network structure beyond their immediate neighbors. The term 'self assembly' is borrowed from applications in biology and molecular science; the parallel is brought out in more detail later.

In our model of the seamless, self assembly (SSA) *problem, file size and transmission times are discretized: F consists of $M \geq 1$ equal-size units; link transmission rates are all the same, and, for convenience, are taken to be one file unit per unit of time. Link transmission delays can vary, but are restricted to positive integers; they label the edges of the network. Seamless transmission of F to a client at the origin server is automatic, so for convenience, but without loss of generality, we consider only problems in which the origin server is not a client node.*

This paper provides an allocation algorithm and a distributed (or network) self-assembly protocol for seamless self-assembly of files at minimum achievable storage cost, i.e., with a minimum total number of file units cached in the nodes of the network. To realize this "perfect" retrieval performance, i.e., stream-like downloading with no initial delay, clients must be prepared to re-sequence segments of a file received out of order. Thus, if the file is a video, for example, the requesting node must experience an initial delay during which some portion of the video is buffered before streaming in the usual sequential sense can begin.

Seamless self-assembly is based on given, typically very simple, finite-state communication protocols at the nodes of the network, as described in detail in the next section. In the distributed setting the input to these protocols is confined to information available locally, and that available from immediate (single-hop) neighbors. The self-assembly process is started by requests issued by the clients wanting the file, which in turn initiates a downloading of the file segments cached at the corresponding client nodes; thereafter, the network autonomically schedules file-unit transmissions so that the remainder of the requested file assembles seamlessly at the client nodes. With self-assembly protocols given, an instance of the SSA problem is an edge-labeled network, a file size M, the identity of the origin-server node, and the subset of client nodes. The problem is to assign subsets of the file $F = \{1, \ldots, M\}$ to the node caches so that seamless downloading is achieved (F is fully assembled at time M at each client) at minimal storage cost, i.e., the total cache occupancy has the minimum achievable size. While this problem has intrinsic interest, there are many obvious extensions to be discussed briefly in the conclusions section. Just as an example, removing the subset of clients from the above instance gives a seemingly much more difficult problem in which cache assignments must be such that, no matter which nodes have clients, seamless self assembly is achieved at each, and solutions must have minimal cost.

This paper focuses on the SSA problem in tree networks, where file-unit transmissions are in the root-to-leaf direction, the root is the origin server for F and all client requests are made simultaneously at time 0. Downloading a unit from any node to its downstream neighbors can be done simultaneously in parallel. Initially, to keep our focus on core issues, we make the *small-network* assumption: The problem instance is such that, in a minimal cost solution, all clients are less than $M/2$ time units away from the root: This condition guarantees that

This work was supported by NSF Grants No. 0117738 and 0092113.

Andreas Constantinides is with the Electrical Engineering Department, Columbia University. Email: acc40@columbia.edu

E. G. Coffman, Jr., is with the Electrical Engineering Department, Columbia University. Email: egc@ee.columbia.edu

[1]An earlier version of this work was presented at the 2^{nd} *Multidisciplinary International Conference on Scheduling: Theory and Applications*, New York, 2005.

at least one unit will be downloaded to every client from the root node. By the self-assembly protocol presented in Section III, this assumption implies that the total extra cache content required for minimal-cost solutions along any path will be less than the file size. In Section V, we remove the small-network assumption and modify our results to apply to the general case.

In the remainder of the paper, δ_v denotes the transmission delay between non-root node v and its parent; $\delta_r \equiv 0$ for the root r. Let V_* be the set of nodes with clients, or with descendants that have clients. We show in Section IV that $\sum_{v\in V_*} 2\delta_v$ is a lower bound to the total cache size needed for seamless self assembly. It is the value of this bound that defines the term *minimal cost*. This lower bound is achieved by the self assembly protocol and cache assignment algorithm for the SSA problem given in Section IV.

II. Motivation and Related Work.

The caching concept has been widely applied, both to computer systems [2], [3] and to the Internet [4], [5], [6], as caches are among the simplest and most effective ways to reduce access times. One of the challenges that web caching currently faces is accommodating the increase in the average size of web pages created primarily by the addition of multimedia files. The number of requests for these larger objects can be expected to increase as more and more users switch to high-speed internet access. Unfortunately, the replacement policies of most currently implemented caching schemes, such as Least-Recently-Used (LRU), are oblivious to file sizes. This leads to huge files in the cache, and consequently, a decrease in the hit ratio of the cache. To alleviate this problem, the LRU-threshold algorithm [7] was introduced. By this algorithm, only files up to a specific size can be stored in the cache. The main limitation of this algorithm is that large files have to be fetched from the origin servers each time they are requested. This increases the average download time of requested files, usually referred to as average access latency.

Recently, the concept of partial caching [8] was introduced to overcome some of the disadvantages of LRU and LRU-threshold. The main idea of partial caching is to store an initial part of the file in the cache and to obtain the rest of the file from the origin server. In this way, the cache hit ratio increases and the average file access latency decreases. The paradigm of seamless downloading grew out of the partial caching concept. Another method to improve the performance of isolated caches is cache co-operation, (see [5], Chapter 9, for a summary of such schemes). Co-operative caching schemes effectively 'increase' the size of an isolated cache [9], [10], and hence they also decrease the access latency.

The problem of how to distribute content efficiently on a network in order to optimize different performance measures has been widely studied. In [11], [12], [13], the authors deal with the question of how to best place a fixed number of caches (web proxies) among potential sites in the network in order to optimize different parameters. In [14], the authors calculate the number of caches needed in order to minimize the overall cost of content distribution. In all of these papers, it is assumed that the requested content is replicated at all the caches. In our work, full replication is not needed, but all nodes are available as caches for content distribution.

The network under consideration in this paper is a set of co-operative caches (in our case they are arranged in a hierarchical structure [16], but they could just as well be arranged in a distributed structure [17]), and the goal is to have seamless downloading at all client nodes with minimal storage cost. Note that, without the use of partial caching, we would have to store the whole file at each client node in order to achieve the minimum achievable QoS guarantees mentioned earlier. However, with the use of partial caching, we are able to achieve seamless downloading by storing just a small fraction of the file at each node.

BitTorrent© (www.bittorrent.com) represents a popular class of systems in which file segments are scattered amongst the nodes of an overlay network and can be assembled by clients in a manner similar to that modeled in this paper. However, such systems are more complicated in that they have global 'tracking' mechanisms; they have more elaborate (and dynamic) request mechanisms, i.e., they do not adhere to seamlessness or to elementary self-assembly protocols; they implement systems of incentives, etc. (See [15] for an approximate analysis of BitTorrent-like systems.) The models studied here contribute to the theoretical underpinnings of file assembly systems operating under self-assembly protocols and seamless transmission constraints.

In the large literature dealing with models of network congestion, file downloading times become service times within a stochastic service (e.g., queueing) system. These studies bring out a strong motivation for seamless self assembly. For, as is well known, expected waiting times and numbers of waiting requests normally grow with the variance of service times. Thus, the zero-variance, constant downloading times of seamless transmission of a given file can be expected to reduce classical measures of congestion in the systems being modeled.

Finally, we mention the intriguing parallels between our self-assembly process and those in biocomputation at nanoscale, where files are the structures/crystals created by distributed growth processes, the node structures play the role of templates, and the file units are the elementary particles/molecules of the growth processes. The initial distribution of cache contents can be thought of as the stoichiometry of the reactants in, say, DNA-based computing. See Adleman et. al. [18] for an introduction to combinatorial self-assembly problems in molecular science; the paper also contains many references to related research in this field.

III. The SSA Problem

Recall that δ_v denotes the integer transmission delay between the non-root node v and its parent $p(v)$ in a given tree T. Thus, if node $p(v)$ starts transmitting a unit of the file to node v at time t, then node v starts receiving it at time $t+\delta_v$ and terminates the download at time $t+\delta_v+1$. The file unit that node v can transmit downstream is either an element of its cache or has just been received from upstream.

The subsets of F stored at the node caches in order to achieve seamlessness are denoted by $S_v = \{S_v(1), \ldots, S_v(k)\}$ ($k = |S_v|$), $v \in T$. When a node v supplies S_v to clients at v, it requires one time unit per file unit in a process that begins immediately. When we say that v receives a unit from upstream at time t, we mean that the reception is complete at time t and took place at v in the interval $(t-1, t]$. The *reception* sequence for a client node v specifies the sequence of file units it receives from its cache and from upstream; a reception sequence for a non-client node specifies only the units it receives from upstream. *Self assembly is* seamless *if the reception sequence of every client node of the tree T is a permutation of* $\{1, \ldots, M\}$. Reception sequences are not defined at the roots of tree instances; instead, the interest there is obviously in the *transmission sequence*, which specifies in each time unit what file unit, if any, the root is transmitting down to one or more of its subtrees.

Our earlier description of self-assembly implied that, apart from request signaling, interior nodes of a tree can perform two functions: they can serve as client nodes as well as forwarding nodes. In order to minimize the total cache content for seamless self assembly, the (local) protocol design objective at every client is to arrange for as much of the file as possible to come from the root r. This objective can be assured by a self-assembly protocol with the following properties.

1) At time 0, every client node v sends a request signal to its upstream neighbor $p(v)$ and begins supplying its own client with its cached units.
2) When a non-root node v receives a request from a node w (necessarily a downstream neighbor), it immediately begins transmitting file units to w, either from v's own cache, if it is non-empty, or by forwarding units being received by v from upstream. If the request is the first one received by v, it is also forwarded immediately to $p(v)$. If node v receives more than one such request simultaneously, it forwards a single request representing all of them.
3) When a node v receives a file unit (necessarily from upstream), it immediately forwards it to all of its downstream neighbors, if any, from which it has received a request, and also supplies it to its own client, if it has one.

Note that the protocol is *on-demand* in that nodes do not participate in the assembly process until activated by requests. There is no effective limit on computational power at the nodes of the network, but we emphasize that no node has any knowledge of network structure beyond its immediate neighbors. We may assume that if a node downloads to two or more neighbors in a given time interval, it must download the same file unit to each; subsequent analysis will show that this assumption incurs no loss in generality (cf. Lemma 3). In principle, a unit arriving at client node v from upstream may have already been transmitted to the client at v. Also, in general, one needs to know what happens under the protocol when a node has both a unit arriving from upstream and local cache units still to be downloaded. Neither of these events needs to be considered here, as they never occur under the cache assignment algorithm for the SSA solution defined later in this section.

Recall that an instance of the SSA problem consists of the edge-labeled tree, T, an integer M denoting the number of units in F, and the subset of nodes that are clients. Without loss of generality we restrict instances to *reduced* trees defined as follows. Let T_v be the subtree of T rooted at v. If T_v consists of just the single node v , it is reduced if and only if v has a client. Otherwise, T_v is reduced if and only if all its subtrees are reduced. Clearly, subtrees not having clients do not participate in the seamless self-assembly problem.

IV. RELATIVELY SMALL NETWORKS.

Let δ_{vw} denote the total transmission delay between nodes $p(v)$ and w (so $\delta_{ww} \equiv \delta_w$). The small-network assumption, which stated that, in a minimal cost solution to the SSA problem, all clients are less than $M/2$ time units away from the root is equivalent to the bound $2\delta_{vw} < M$ for all nodes w in any subtree T_v of the root. The following lemmas and Theorem 1 give lower bounds on cache occupancies that account for the times taken by requests to go upstream from w, and then for file units to travel downstream to w.

Lemma 1. *The cache at client node v in a seamless solution to an instance of the SSA problem on trees must have at least $2\delta_v$ distinct file units.*

Proof: Suppose that v's cache size satisfies $|S_v| < 2\delta_v$. By the small-network assumption, $M > 2\delta_v$, so in a seamless solution, v must receive part of the file from upstream. But it takes $2\delta_v$ time units for v's request to reach $p(v)$ and for a file unit to come back down to v. This implies that there will be one or more of the first $2\delta_v$ time units when the client at v is not receiving a file unit – a contradiction to the assumed seamless self assembly at v.

□

We extend this result to paths from non-clients to clients, as follows.

Lemma 2. *In a reduced instance of the SSA problem on trees, let v be a non-client and consider the path from v to a downstream client node w whose request is among the earliest to arrive at v. In a seamless solution to this SSA problem, the union of the caches in the path from v to w must have at least $2\delta_{vw}$ distinct file units.*

An illustration is shown in Figure 1, where the file size is $M = 11$, and nodes c, d, and e are clients.

Node b receives an earliest request from node d at time $t = 2$, but the earliest time at which node b can start receiving units from upstream is $t = 4$, and these units can start arriving at node d at time $t = 6$. Thus, for example, if the total number of distinct file units in the caches of nodes b and d is at least 6, client node d can have seamless self assembly.

□

Proof of Lemma 2: The argument proceeds as before. By the small-network assumption, the total delay along the path from

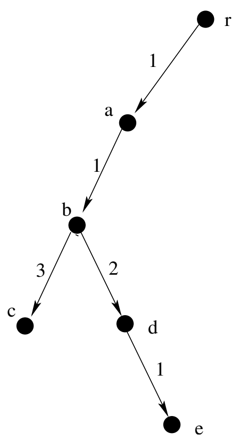

Fig. 1. An illustration for the proofs of Lemma 2 and Theorem 1

$p(v)$ to w must satisfy $2\delta_{vw} < M$. Assume that the union of the caches at the nodes along the path starting with v and ending at w contains fewer than $2\delta_{vw}$ distinct file units. Then by the self-assembly protocol, one or more file units must be brought down to w from nodes upstream of v. But such a file unit cannot start arriving at w until time $2\delta_{vw}$, since w's request is the first, or one of the first, to arrive at v. Then in the first $2\delta_{vw} < M$ time units, there must be at least one time unit during which no file unit is being downloaded to the client at w - a contradiction to the assumed seamlessness of the solution.

□

Theorem 1. *In a seamless solution to a reduced instance of the SSA problem on trees, a lower bound on the total number of file units that must be stored in the caches of the non-root nodes of the tree is twice the sum of the edge transmission delays.*

Proof: We prove a stronger result: For any non-root node v in the given reduced instance, the theorem applies to T_v in isolation. Precisely, a lower bound for the total number of units stored in the caches of the nodes in T_v must be equal to $\sum_{u \in T_v} 2\delta_u$. In order to do this, we define a partition π of T_v into a set of disjoint paths; for each path $\mathcal{P} \in \pi$ we show that the caches of the nodes on $\mathcal{P}$ must store at least $\sum_{u \in \mathcal{P}} 2\delta_u$ distinct file units. The theorem follows immediately.

We define w to be a *closest client* to non-client node u, if w's file request is among the earliest to arrive at u. (Recall that a non-client node u might receive an earliest request from more than one client, but it forwards only one upstream representing all of them.) Consider for example the tree in Figure 1, and assume that nodes a, c, and e are the client nodes. Node e is a closest client to node d, but node b receives two earliest requests at time $t = 3$ from client nodes c and e. Then node b has *two* closest clients: nodes c and e.

Now partition T_v into a set π of disjoint paths such that (i) each contains exactly one client node, and (ii) this single client node is a closest client to all other nodes on the path. (Note that paths may consist of only one node.) Our assumption of reduced trees implies that all leaf nodes in T_v are clients, so that each internal node v has at least one client in its subtree. Thus, every non-client node has at least one closest client and a partition with the desired properties exists. In the case where at least one non-client node has two or more closest clients, such partitions are clearly not unique.

Returning to the earlier example, consider the subtree rooted at node a. There are 3 client nodes in T_a, so partition results in 3 disjoint paths. However, because node b has two closest clients, we can have two possible partitions: (1) $\{a\}, \{c\}$, and $\{b, d, e\}$; and (2) $\{a\}, \{b, c\}$, and $\{d, e\}$. In the former partition, node a has to store 2 distinct units in its cache, node c has to store 6 distinct units in its cache, and the total number of distinct units stored in the caches of nodes b, d, and e must be at least 8. In the latter partition, node a has to store 2 distinct units in its cache, the total number of distinct units stored in the caches of nodes b and c must be at least 8, and the total number of distinct units stored in the caches of nodes d and e must be at least 6.

By Lemma 1, all paths with a single node consist of a client node, say u, with a cache containing at least $2\delta_u$ distinct file units, if u is to have seamless self assembly. Similarly, by Lemma 2, if the client node on a path $\mathcal{P} \in \pi$ of two or more nodes is to achieve seamless self assembly, the total number of distinct file units that must be stored in the caches of the nodes on $\mathcal{P}$ must be at least $\sum_{u \in \mathcal{P}} 2\delta_u$. (Note that in our example, in both partitions a lower bound on the total number of units that have to be stored in the caches of the network is equal to 16, which is twice the sum of the edge transmission delays.)

□

Recall that a solution to the SSA problem is said to have minimal cost if it is seamless and if it achieves the lower bound of Theorem 1, i.e., if for all non-root nodes $v \in T$, the total number of file units cached at the nodes in T_v is equal to $\sum_{u \in T_v} 2\delta_u$.

The following lemma simplifies the self-assembly protocol for the SSA problem.

Lemma 3. *A minimal cost solution for any given tree can be constructed from minimal cost solutions to the subproblems created by removing all but one subtree of the root.*

Proof: One needs to verify that there is a minimal cost solution for which the root never needs to transmit different file units in the same time interval, assuming that minimal cost solutions exist for the subproblems. Choose a subtree of the root with a client closest to the root, and remove the other subtrees. In a minimal cost solution, which we assume exists, let its root transmission sequence be σ. (Because of the closest-client assumption, the sequence σ is as long as that of any other subproblem.) Now add back one of the subtrees and permute the file-unit numbering adopted for its minimal cost solution so that its root transmission sequence is a subsequence of σ aligned at (occupying) the same time units. Continuing the process, we see that permutations can always be selected so that file units downloaded simultaneously to two or more children of the root are always the same.

□

The following cache assignment algorithm yields minimal cost solutions for trees. Furthermore, it assigns exactly $2\delta_v$ file units to each cache.

> THE CYCLIC SHIFT (CS) ALGORITHM. Begin with an arbitrary node of the reduced tree and give it an arbitrary permutation of $1, 2, ..., M$, its *sequence*. While there are still nodes not having been assigned a sequence, pick such a node, say u which has a parent or a child already with its sequence. If the parent w of u has a sequence, then u's sequence is obtained by shifting w's sequence δ_u time units cyclically to the right, and if w is a child of u, then u's sequence is obtained by shifting cyclically w's sequence δ_w to the left.
> The cache at a client node v contains the first $2\delta_v$ units of its sequence. In particular, the ℓ-th unit of v's sequence is assigned to $S_v(\ell)$, $1 \leq \ell \leq 2\delta_v$. For a non-root, non-client node u, let $j > 0$ be the earliest time it receives a request. Then the cache at u contains the $2\delta_u$ units $j + 1$ through $j + 1 + 2\delta_u$ of its sequence. (The $(j + \ell)$-th unit of u's sequence is assigned to $S_u(\ell)$, $1 \leq \ell \leq 2\delta_u$.)

If follows from the CS algorithm that the self-assembly protocol defines a unique on-demand solution to the SSA problem: the reception sequence of a client node v is the sequence assigned to it by the CS algorithm. It consists of its cache units followed by the units it receives from upstream. The reception sequence of a non-client node v contains just those units of its sequence which it receives from upstream, namely the units in positions $2\delta_v + j + 1, \ldots, M - j$.

Let t_v^* denote the time at which node v first receives a request (including the one from its own client, if it has one), and let $v_1, \ldots, v_c$ be the downstream neighbors of v. We first note that for a given set of file requests, the times t_v^*, that are used in the CS algorithm, are all easily computable recursively starting at the bottom of the reduced tree and using,

$$t_v^* = \begin{cases} 0 & \text{if } v \text{ is a client node,} \\ \min\limits_{1 \leq i \leq c} \{t_{v_i}^* + \delta_{v_i}\} & \text{otherwise} \end{cases} \tag{1}$$

We should also note here the stopping problem: How does one discontinue the transmission of units once the initial objectives have been met, i.e., each node has the reception sequences described above? One way would be to enable the nodes to count. In that case, if a node v keeps track of the time $t_u^* + \delta_u$ at which it has received a request from its child u, it can start transmitting units immediately and stop transmitting units to u as soon as it has transmitted exactly $M - 2(t_u^* + \delta_u)$ units, (i.e., at time $M - t_u^* - \delta_u$). If we are not concerned about the transmission of extra units that are not necessary for seamlessness at any client, then we can allow nodes to transmit units as long as they receive units from upstream. Of course, in that case the reception intervals of interest at each node would be the ones outlined in the previous paragraph.

Before proving that solutions under the CS algorithm and the self-assembly protocol are minimal cost solutions, we illustrate the CS algorithm in action.

Example 1: An example is shown in Figure 2 for a file of 12 units and the tree shown, which has b, c, and d as client nodes. Let CS start by assigning the sequence $1, \ldots, 12$ to node b. Then, by the CS algorithm, we obtain the cache allocations $S_a = \{3, 4\}$, $S_b = \{1, 2\}$, $S_c = \{10, 11, 12, 1, 2, 3\}$, and $S_d = \{12, 1, 2, \ldots, 5\}$, as can be seen from the reception sequences shown in Figure 3 for all nodes. The underlined units in Figure 3 are the cached units.

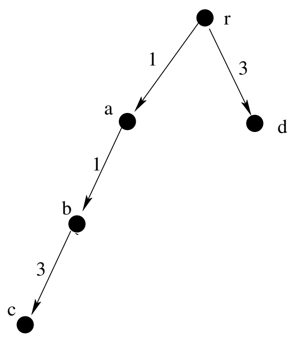

Fig. 2. The tree under consideration for the SSA problem

a:				5	6	7	8	9	10	11	12	
b:	$\underline{1}$	$\underline{2}$	3	4	5	6	7	8	9	10	11	12
c:	$\underline{10}$	$\underline{11}$	$\underline{12}$	$\underline{1}$	$\underline{2}$	$\underline{3}$	4	5	6	7	8	9
d:	$\underline{12}$	$\underline{1}$	$\underline{2}$	$\underline{3}$	$\underline{4}$	$\underline{5}$	6	7	8	9	10	11

0 1 2 3 4 5 6 7 8 9 10 11 12 → time, t

Fig. 3. The reception sequences for the SSA problem when b, c, and d are clients for the file

The example illustrates an important point: In general, flows of units through some node v to one of its subtrees, say T'_v, may be already in progress when a request arrives from one of v's other subtrees, say T''_v. For the CS algorithm to work, it is clear that the units that have already been sent to downstream nodes in T'_v must be cached at one or more downstream nodes in T''_v. This is illustrated at node b in Figure 1. Node b starts receiving units from upstream at $t = 2$, but only receives the request from node c at $t = 3$. Thus, the first unit that node b receives from upstream cannot be sent down to node c. However, that unit, viz. 3, along with the cache at node b, is cached according to the CS algorithm at node c, thus ensuring seamless self-assembly at all clients and a minimal cost solution.

□

Theorem 2. *Under the self-assembly protocol for trees, the CS algorithm yields a minimal cost solution to the SSA problem with the small-network assumption.*

Proof: By Theorem 1, the solution to the SSA problem produced by the CS algorithm is clearly minimal cost if it is seamless, since by definition each non-root node v caches exactly $2\delta_v$ file units. Thus, it remains to prove seamlessness.

Cyclic shifts of permutations yield permutations, so it follows immediately from the definition of the CS algorithm that all client nodes have reception sequences that are permutations of $\{1, \ldots, M\}$. The CS solution will be seamless if the following holds by the self-assembly protocol: Let v be a node with an upstream (non-root) node w. Then a unit in v's reception sequence but not in its cache either appears δ_v time units earlier in w's reception sequence, or it appears in w's cache.

Note that the earliest that the parent w can receive a request from v is $t_v^*+\delta_v$ and by the definition of t_w^*, we have $t_w^* \leq t_v^*+\delta_v$, i.e., w begins transmitting to some downstream neighbor, or to its own client if it has one, no later than the time it receives a request from v. Note also that in the time interval t_w^*+1 through $t_w^*+2\delta_w+1$, node w transmits units from its cache. A unit in v's reception sequence but not in v's cache must appear in v's reception sequence no earlier than $t_v^* + 2\delta_v + 1 \geq t_w^* + \delta_v + 1$. It follows from the shifts made in successive reception sequences by the self-assembly protocol that the unit in position $t_v^*+2\delta_v+i$ of v's reception sequence must then appear in position $t_v^*+\delta_v+i \geq t_w^*$ of w's reception sequence, or in w's cache, and the theorem is proved.

□

V. The General Case

Although some care is required, it is not difficult to adjust the arguments and results to the general case with no constraint on file size relative to tree structure. Without the small-network assumption, there will in general be nodes, the root in particular, that are so far away from one or more clients that the these too-distant clients can not receive units from the root in a seamless solution.

If we eliminate the small-network constraint for the SSA problem, Lemmas 1 and 2 of the paper still apply with only a slight modification.

Lemma 1′. *The cache at client node v in a seamless solution to an instance of the SSA problem on trees must have at least $\min\{M, 2\delta_v\}$ distinct file units.*

Recall that δ_{vw} denotes the total transmission delay between nodes $p(v)$ and w (so $\delta_{ww} \equiv \delta_w$). We extend the result of Lemma 1′ to paths from non-clients to clients, as follows.

Lemma 2′. *In a reduced instance of the SSA problem on trees, let v be a non-client and consider the path from v to a downstream client w whose request is among the earliest to arrive at v. In a seamless solution to this instance of the SSA problem, the union of the caches in the path from v to w must have at least $\min\{M, 2\delta_{vw}\}$ distinct file units.*

The proofs of Lemmas 1′ and 2′ are omitted since they are very similar to the proofs of Lemmas 1 and 2. Now, denote by V_+ the set of all non-root nodes in the reduced tree which first receive a request before time $M/2$, i.e., $t_v^* < M/2$. Each node in V_+, by its proximity to at least one client, must participate in the seamless downloading of the file. Then, the following analog of Theorem 1 holds.

Theorem 3. *In a seamless solution to a reduced instance of the SSA problem on trees, a lower bound on the total number of file units that must be stored in the caches of the non-root nodes of the tree is equal to*

$$\sum_{v \in V_+} \min\{2\delta_v, M - 2t_v^*\} \tag{2}$$

Proof: Consider any non-client node v of the reduced tree which first receives a request at or after time $M/2$. Even if v immediately starts transmitting units to the client, say u, from which it has received this request, node u can not begin receiving these units until after time M. We conclude that no units should be stored in the cache of node v. Thus, it remains to deal with the nodes in V_+.

As before, we can partition the nodes in V_+ into a set π of disjoint paths such that (i) each contains exactly one client node, and (ii) this single client node is the closest client to all other nodes on the path. Our assumption of reduced trees implies that all leaf nodes in V_+ are clients, so that each internal node $v \in V_+$ has at least one client in its subtree. Thus, every non-client node has at least one closest client and a partition with the desired properties exists. As before, in the case where at least one non-client node has two or more closest clients, such partitions are not unique.

By Lemma 1′, all paths with a single node consist of a client node, say u, with a cache containing at least $\min\{M, 2\delta_u\}$ distinct file units, if u is to have seamless self assembly. Note that for client nodes $t_u^* = 0$, so it follows that $M = M - 2t_u^*$, and $\min\{M, 2\delta_u\} = \min\{M - 2t_u^*, 2\delta_u\}$. Similarly, by Lemma 2′, if the client node on a path $\mathcal{P} \in \pi$ of two or more nodes is to have seamless self assembly, the total number of distinct file units that must be stored in the caches of the nodes on $\mathcal{P}$ must be at least $\min\{M, \sum_{u \in \mathcal{P}} 2\delta_u\}$. We show below that, for paths of two or more nodes, $\min\{M, \sum_{u \in \mathcal{P}} 2\delta_u\} = \sum_{u \in \mathcal{P}} \min\{2\delta_u, M - 2t_u^*\}$, from which the theorem follows immediately.

Let x be the node in $\mathcal{P}$ closest to the root. Observe that, because all nodes in $\mathcal{P}$ have the same closest client, the following holds: $t_x^* = \sum_{u \in \mathcal{P}-x} \delta_u$. Thus,

$$\begin{aligned} \min\{M, \sum_{u \in \mathcal{P}} 2\delta_u\} &= \sum_{u \in \mathcal{P}-x} 2\delta_u + \min\{M - \sum_{u \in \mathcal{P}-x} 2\delta_u, 2\delta_x\} \\ &= \sum_{u \in \mathcal{P}-x} 2\delta_u + \min\{M - 2t_x^*, 2\delta_x\} \end{aligned} \tag{3}$$

Consider now the downstream neighbor y of x in $\mathcal{P}$ (which exists since we are considering paths $\mathcal{P}$ with at least 2 nodes). Since $2t_x^* < M$, and $t_x^* = t_y^* + \delta_y$, it follows that $2\delta_y < M - 2t_y^*$. Hence, we can see that $2\delta_y = \min\{2\delta_y, M - 2t_y^*\}$. Similar arguments show that for each $u \in \mathcal{P} - x$, $2\delta_u = \min\{2\delta_u, M - 2t_u^*\}$. Thus,

$$\sum_{u \in \mathcal{P}-x} 2\delta_u = \sum_{u \in \mathcal{P}-x} \min\{2\delta_u, M - 2t_u^*\} \tag{4}$$

Substituting (4) in (3), we get

$$\min\{M, \sum_{u\in\mathcal{P}} 2\delta_u\} = \sum_{u\in\mathcal{P}} \min\{2\delta_u, M - 2t_u^*\} \quad (5)$$

And the theorem is proved.

□

In the general case, a solution to the SSA problem is said to have minimal cost if it is seamless and if it achieves the lower bound of Theorem 3, i.e., $\sum_{v\in V_+} \min\{2\delta_v, M - 2t_v^*\}$. A very slightly modified version of the CS algorithm gives a minimal cost solution for the SSA problem. The modification must be such that it concentrates in allocating file units only to caches of nodes in V_+, (exactly $\min\{2\delta_v, M - 2t_v^*\}$ units to each v in V_+), since all nodes receiving a first file request after time $M/2$ cannot participate in the seamless self assembly process. Also, a result of the new algorithm will be that some nodes in V_+ might have to store fewer units than twice the delay with their parent node.

> THE GENERAL CYCLIC SHIFT (GCS) ALGORITHM. Begin with an arbitrary node of the reduced tree and give it an arbitrary permutation of $1, 2, ..., M$, its *sequence*. While there are still nodes not having been assigned a sequence, pick such a node, say u which has a parent or a child already with its sequence. If the parent w of u has a sequence, then u's sequence is obtained by shifting w's sequence δ_u time units cyclically to the right, and if w is a child of u, then u's sequence is obtained by shifting cyclically w's sequence δ_w to the left.
> For a client node v, if $2\delta_v < M$, the cache at v contains the first $2\delta_v$ units of its sequence. In particular, the ℓ-th unit of v's sequence is assigned to $S_v(\ell)$, $1 \le \ell \le 2\delta_v$. If $2\delta_v > M$, then v only stores M units in its cache, (i.e., the whole file F).
> For a non-root, non-client node u, let $j > 0$ be the earliest time it receives a request from downstream. If $j \ge M/2$, then node u does not participate in the file downloading. If $j < M/2$, the cache at u contains the $\min\{2\delta_u, M - 2j\}$ units $j + 1$ through $j + 1 + \min\{2\delta_u, M - 2j\}$ of its sequence. (The $(j + \ell)$-th unit of u's sequence is assigned to $S_u(\ell)$, $1 \le \ell \le \min\{2\delta_u, M - 2j\}$.)

The reception sequence of a client node v is the sequence assigned to it by the GCS algorithm. It consists of its cache units followed by the units it receives from upstream. The reception sequence of a non-client node v contains just those units of its sequence which it receives from upstream, if any.

Note again that we can use (1) in order to efficiently calculate the time t_v^* at which node v of the reduced tree first receives a request. A systematic way of doing this would be by a post-order walk on the reduced tree.[2] Assuming that a sequence is already assigned to each node by the GCS algorithm, the calculation of the t_v^* values and the assignments of units to caches can be done at the same time during a post-order walk on the tree.

Also note that the stopping problem as defined earlier still exists. On top of that, we must be concerned with the problem of forwarding requests to nodes so far upstream that downloading in a seamless solution is impossible. If nodes had synchronized clocks, then request forwarding could be stopped after time $M/2$; but synchronism might violate the assumed absence of any global control mechanism beyond that which establishes the initial state. Alternatively, given the nature of the allocation of the GCS algorithm, one could require that any node v with fewer than $2\delta_v$ units allocated to its cache not forward a request upstream, because the delay of the parent node of v and the client in the subtree of v is more than $M/2$ units. Finally, hop counts kept in the headers of requests could be another mechanism for eliminating useless forwarding. The example below illustrates the GCS algorithm at work.

Example 2: The example shown in Figure 4 has a file of 5 units and the tree shown with b, e, and f as client nodes. Let GCS start by assigning the sequence $1, \dots, 5$ to node a. Then, by the GCS algorithm, we obtain the cache allocations $S_a = \{2, 3\}$, $S_b = \{5, 1\}$, $S_c = \emptyset$, $S_d = \{1\}$, $S_e = \{2, 3, 4, 5\}$, and $S_f = \{1, 2, ..., 5\} = F$ as can be seen from the reception sequences shown in Figure 5 for all nodes. The underlined units in Figure 5 are the cached units.

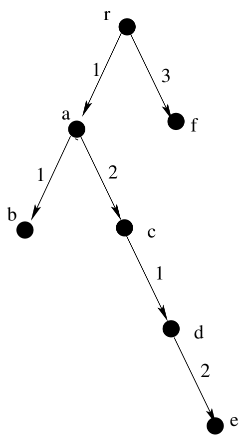

Fig. 4. The tree in Example 2

	0–1	1–2	2–3	3–4	4–5
a:				4	
b:	$\underline{5}$	$\underline{1}$	2	3	4
c:					
d:					
e:	$\underline{2}$	$\underline{3}$	$\underline{4}$	$\underline{5}$	1
f:	$\underline{3}$	$\underline{4}$	$\underline{5}$	$\underline{1}$	$\underline{2}$

time, t: 0 1 2 3 4 5

Fig. 5. The reception sequences when b, e, and f are clients

Note that node d stores only 1 unit of the file, and node c

[2]Recall that a post-order walk of a tree T first traverses (recursively) the left subtree of T in post-order , then traverses the right subtree of T in post-order, and then visits the root of T.

does not participate in the file transmission as it receives an earliest request at time $t = 3$. Also note that client node f caches the whole file.

$\square$

Theorem 4. *Under the self-assembly protocol for trees, the GCS algorithm yields a minimal cost solution to the SSA problem.*

Proof: By Theorem 3, the solution to SSA produced by the GCS algorithm is clearly minimal cost if it is seamless, since by definition each non-root node v in V_+ caches exactly $\min\{2\delta_v, M - 2t_v^*\}$. It remains to prove seamlessness. Since this can easily be done by arguments very similar to the ones used in the proof of Theorem 2, these details are omitted.

$\square$

VI. Conclusions and Directions for Further Research

This section considers several generalizations of the basic SSA problem, which is conveniently called CLIENT in this section. Two of these are singled out for further discussion within the simplification of the small-network assumption: the ANY-CLIENT and CLIENT-RELEASE problems described in the two subsections to follow. The lower bound of Theorem 1 applies to solutions of both problems. However, as we will see this lower bound is not generally achievable. We leave the detailed study of the more elaborate solutions to ANY-CLIENT and CLIENT-RELEASE to a future paper currently in preparation.

A. The ANY-CLIENT Problem

Suppose that we now remove the subset of clients from the problem instance, but require that, no matter which nodes are chosen as clients, a minimal cost solution be obtained. Under the distributed self-assembly protocol for the CLIENT problem, such solutions cannot be found in general. We show below that the lower bound of Theorem 1 is not generally achievable for the ANY-CLIENT problem.

Example 3: Consider the ANY-CLIENT problem on a path of length 5, where the one-way transmission delay from any non-root node v to its parent is 1. For convenience, label the nodes $1, 2, \ldots, 5$, with node 1 being the child of the root, r.

Each node has to cache the minimum number of file units needed to cover the two-way communication delay with its parent. Since node 5 might be the only client for the file, these $2\delta_v$ units have to be disjoint. Let S_v denote the set cached at node v, and without loss of generality do the following allocation: $S_1 = \{1, 2\}$, $S_2 = \{3, 4\}$, $S_3 = \{5, 6\}$, $S_4 = \{7, 8\}$, and $S_5 = \{9, 10\}$.

Assume that at time $t = 0$, the clients at nodes 2, and 5 request the file. (This is a case where two client nodes are separated by two non-client nodes.) We will now show that with the given allocation scheme, seamless self-assembly is not possible at both clients.

The reception sequence at node 2 has to begin with the units in S_2, followed by the units in S_1. The reception sequence at node 5 has to begin with the units in S_5, followed by the units in S_4, followed by the units in S_3. In order to achieve seamless self-assembly, node 2 has to receive the set of units $\{5, 6, \ldots, M\}$ in the time units $5, 6, \ldots, M$. Now, note that one of the units, say u, in S_1, is downloaded at node 2 before the request from node 5 arrives at 2. So if 5 is to achieve seamless self-assembly, unit u has to be brought from node 1 some time later so that it can go down to node 5. But if unit u is downloaded again at node 2, then seamless self-assembly will be impossible at node 2.

$\square$

B. The CLIENT-RELEASE Problem

Another problem that begs consideration is CLIENT-RELEASE in which the times the client nodes make their requests are allowed to be arbitrary integers which become part of the problem instance. Thus, an instance of the CLIENT-RELEASE problem consists of the edge-labeled tree, an integer M denoting the number of units in F, the subset of nodes that are clients and their respective release times, (i.e., the times the client nodes make their requests), which are allowed to be arbitrary integers. The following example on a path shows the complexity of the CLIENT-RELEASE problem.

Example 4: Consider a path of length 3, where the one-way transmission delay from any non-root node v to its parent is 1, and all three nodes are clients. For convenience, label the nodes $1, 2, 3$, with node 1 being the child of the root, r. Denote the release time of node v by t_v, and denote the time at which node v first receives a request (including the one from its own client) by t_v^*. (Note that because the release times are allowed to be arbitrary integers, t_v^* does not necessarily have to be equal to t_v.) Let $M = 7$, $t_3 = 0$, $t_1 = 5$, and $t_2 = 6$. Thus, it follows that $t_3^* = 0$, $t_2^* = 1$, and $t_1^* = 2$. Each node has to cache the minimum number of file units needed to cover the two-way communication delay with its parent, and given the above release times we see that these $2\delta_v$ units have to be disjoint. Let S_v denote the set cached at node v, and without loss of generality do the following allocation: $S_1 = \{1, 2\}$, $S_2 = \{3, 4\}$, and $S_3 = \{5, 6\}$. The reception sequence at node 3 has to begin (at time $t = 0$) with the units in S_3, followed by the units in S_2, followed by the units in S_1, followed by unit 7 that originates from the root. Thus, it is easy to see that client node 3 can have seamless self-assembly. We will now show that with the given allocation scheme, seamless self-assembly is impossible at both clients 1 and 2.

Now, at time $t = 5$ the reception sequence at node 1 has to begin with the units in $S_1 = \{1, 2\}$ followed by 5 units originated from the root, and at time $t = 6$ the reception sequence at node 2 has to begin with the units in $S_2 = \{3, 4\}$ followed by 5 units sent to 2 from 1. Remember at this point that a node can only send the same unit to its children and to its own client if it has one. Also note that the time interval during which node 1 will receive 5 units from the root, (this interval is the time units $8, 9, \ldots, 12$), coincides with the time interval it has to send 5 units to node 2. In order for node 1 to have seamless self-assembly, it has to receive units $\{3, \ldots, 7\}$ from the root. However, in order for node 2 to have seamless self- assembly, node 1 has to send to node 2 in the same time interval the units $\{1, 2, 5, 6, 7\}$. Thus, we see that it is impossible for both clients to have seamless self-assembly with the given allocation scheme.

$\square$

C. Final remarks

Directions for yet further research, staying within the domain of this paper include the obvious generalizations of our assumptions. For example, we are currently working on solutions to ANY-CLIENT for trees under the more relaxed distributed protocols of flooding paradigms.

Problems like CLIENT, ANY-CLIENT, and CLIENT-RELEASE serve as tests of the complexity of many more general problems which are our eventual objective. Another interesting question that we are pursuing is the relation between streaming and seamlessness: From the allocations in the set of minimal cost solutions, which allocation also minimizes the maximum delay until streaming can begin at each client? Multiple-file problems, trees where file units can also go upstream, structures more general than trees, and non-homogeneous link transmission rates are among the many directions for future research in seamless self assembly that are worth pursuing.

REFERENCES

[1] E. G. Coffman, A. Constantinides, D. Rubenstein, B. Shepherd, and A. Stavrou, "Content distribution for seamless transmission," *SIGMETRICS Perform. Eval. Rev.*, vol. 32, no. 2, pp. 31–32, 2004.

[2] A. J. Smith, "Cache memories," *ACM Comput. Surv.*, vol. 14, no. 3, pp. 473–530, 1982.

[3] S. A. Przybylski, *Cache and memory hierarchy design: a performance-directed approach.* San Francisco, CA, USA: Morgan Kaufmann Publishers, Inc., 1990.

[4] J. Wang, "A survey of web caching schemes for the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 5, pp. 36–46, 1999.

[5] M. Rabinovich and O. Spatschek, *Web caching and replication.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] S. V. Nagaraj, *Web Caching and Its Applications.* Norwell, MA, USA: Kluwer Academic Publishers, Inc., 2004.

[7] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, "Caching proxies: Limitations and potentials," in *Proceedings of 4th International World Wide Web Conference*, Boston, USA, December 1995, pp. 119–133.

[8] S. Sen, J. Rexford, and D. F. Towsley, "Proxy prefix caching for multimedia streams," in *Proceedings of IEEE INFOCOM*, New York, March 1999, pp. 1310–1319.

[9] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. R. Karlin, and H. M. Levy, "On the scale and performance of cooperative web proxy caching," in *Symposium on Operating Systems Principles*, 1999, pp. 16–31.

[10] P. Rodriguez, C. Spanner, and E. W. Biersack, "Analysis of web caching architectures: hierarchical and distributed caching," *IEEE/ACM Trans. Netw.*, vol. 9, no. 4, pp. 404–418, 2001.

[11] B. Li, M. J. Golin, G. F. Italiano, and X. Deng, "On the optimal placement of web proxies in the internet," in *Proceedings of IEEE INFOCOM*, New York, March 1999, pp. 1282–1290.

[12] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 568–582, 2000.

[13] K. Li and H. Shen, "Optimal placement of web proxies for tree networks," in *Proceedings of the 2004 IEEE Conference of e-Technology, e-Commerce and e-Service*, March 2004, pp. 479–486.

[14] I. Cidon, S. Kutten, and R. Soffer, "Optimal allocation of electronic content," *Computer Networks*, vol. 40, no. 2, pp. 205–218, 2002.

[15] D. Qiu and R. Srikant, "Modeling and performance analysis of bittorrent-like peer-to-peer networks", in *Proceedings of ACM Sigcomm*, Portland, OR, Aug 2004.

[16] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," in *USENIX Annual Technical Conference*, 1996, pp. 153–164.

[17] D. Povey and J. Harrison, "A distributed internet cache," in *Proceedings of the 20th Australasian Computer Science Conference*, Sydney, Australia, February 1997.

[18] L. Adleman, Q. Cheng, A. Goel, M.-D. Huang, D. Kempe, P. M. de Espanes, and P. W. K. Rothemund, "Combinatorial optimization problems in self-assembly," in *STOC '02: Proceedings of the thiry-fourth annual ACM Symposium on Theory of Computing.* New York, NY, USA: ACM Press, 2002, pp. 23–32.