On the Steady-State of Cache Networks

Elisha J. Rosensweig
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003–9264
Email: elisha@cs.umass.edu

Daniel S. Menasche
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003–9264
Email: sadoc@cs.umass.edu

Jim Kurose
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003–9264
Email: kurose@cs.umass.edu

Abstract—**A shift has recently begun taking place regarding the manner in which researchers are thinking about networking in the Internet. In addition to the traditional host-to-host communication that has endured for more than four decades, many researchers have begun to focus on *Content Networking* - a networking model in which host-to-content interaction is the norm. A central component of such an architecture is a large-scale interconnected caching system. To date, very little is understood about the way these networks of such caches behave and perform.**

In this work, we demonstrate that certain cache networks are non-ergodic in that their steady-state characterization depends on the initial state of the system. We then establish several important properties of cache networks, in the form of three sufficient conditions for a cache network to be ergodic. Each property targets a different aspect of the system - topology, admission control and cache replacement policies. Perhaps most importantly we demonstrate that cache replacement can be grouped into equivalence classes, such that the ergodicity (or lack-thereof) of one policy implies the same property holds for all policies in the class.

I. INTRODUCTION

A shift has recently begun taking place regarding the manner in which researchers are thinking about networking in the Internet. In addition to the traditional host-to-host communication that has endured for more than four decades, many researchers have begun to focus on Content Networking - a networking model in which content is addressable and host-to-content interaction is the norm [1][2][3][4][5][6][7]. This transition is a culmination of many changes in end-user demands (such as the astounding growth of P2P systems), a focus on content distribution, and the increasing mobility of network elements.

With content accessibility in the spotlight, caches are poised to become central components of any Content Networking architecture. Storing large chunks of popular content at several locations in such a network can greatly reduce load on servers, congestion in the network and service delays at the end-user. While many traditional systems employ caching as a means to improve performance (see [8][9][10][11][12] for a small sample of such discussions), dealing with the analysis and management of Internet-scale networks of uncoordinated caches, termed here *Cache Networks* and abbreviated "CNs", is a relatively uncharted field.

Analyzing these systems is a daunting task. Even for a single stand-alone cache, referred to here as an *isolated* cache,

exact models of system behavior for common systems [13] are intractable due to state explosion, causing some to seek approximation tools for single-cache systems [14]. The situation is complicated further once we place many such caches in a network, since their presence introduces considerable changes to the endogenous flows in the system [15], by applying a *selective filter* on these flows - a request for content is only forwarded to a subsequent cache if it is not found in the current cache where the request has arrived [11][16]. As a result, existing analytical work in this field has generally been limited to small networks, specific topologies, and homogeneous cache management policies. Furthermore, models for similar systems do not capture the complexity of CNs. CDNs employ many caches as part of their architecture, but at the core they are still client-server systems that redirect requests to where content is known to be cached, whereas we are considering systems where caches have loose or no coordination amongst them, and cache contents change as content is forwarded among the network caches. System analyses in the P2P literature also fall short of supplying us with sufficient models, due to the limited role overlay networks play in *forwarding* content.

In this paper, we take first steps in understanding the behavior of cache networks, focusing on factors that impact the steady-state of content occupancy, which directly impacts performance. Analytical models for caching systems usually take into account the cache capacity (i.e., how much storage is at its disposal), the topology of the cache network (e.g., hierarchical), the cache-management policies and the exogenous request arrival rates per-file at each cache [10][17][18][19]. Absent from this list is the *initial state* of the system - the files stored in each cache when the system is initialized. The fact that the initial state is ignored reflects an (explicit or implicit) intuition that these systems are *ergodic*, i.e., once the caches have undergone a "warmup" phase in which they are populated based on user demand, the initial state becomes irrelevant to the system behavior in the long run. In this paper we consider this assumption and its validity as a function of various system properties. Our work here is an attempt to begin building an "analytical toolkit" for analyzing the performance of cache networks. Just as queueing theory was central for understanding the behavior of packet-switching networks, we believe that developing such a set of analytical tools is crucial if these cache networks are to be understood.

The major contributions of this paper are the following:

- We present two examples of non-ergodic CNs, in the sense that different content placed initially at the caches will lead to different steady, long term behavior. In both examples, the observed behavior arises only when the caches are interconnected.
- We establish several important properties of CNs, in the form of three sufficient conditions for the CN system to be ergodic. Each property targets a different aspect of the system - topology, admission control and cache replacement policies.
- With regard to replacement policies, we demonstrate that these can be grouped into "equivalence classes", such that the ergodicity (or lack-thereof) of one policy implies the same property holds for all replacement policies in the class.

The structure of this paper is as follows. We begin in §II by presenting the system model used throughout this paper. Then, in §III, we present two examples of non-ergodic cache networks, in that the initial state determines the steady-state that the system converges to, with an impact on system performance. In §IV we formulate and prove two theorems regarding system ergodicity, which relate to network topology and admission control. Then, in §V we outline a class of cache replacement policies for which the system is always ergodic. We also identify equivalence classes of replacement policies such that the ergodicity (or lack-thereof) of one policy implies the same holds for other policies in that class. We then review related work in §VI, discuss of some of our model assumptions in §VII, and conclude the paper with a summary and discussion of future work in §VIII.

II. MODEL AND NOTATION

A. System Components and Operation

Basic model. We begin by describing the system of interest in this paper - cache networks (CNs). A summary of the notation that follows is presented in Table I. Our model follows common practices (see for example [10], [20], [21] and [17]).

Let $G = \langle V, E \rangle$ be a finite network comprised of nodes $V = \{v_1, ... v_m\}$ and edges $E \subseteq V \times V$. Each node corresponds to a *cache-router* element - a router augmented with short-term storage capabilities, such that content that is forwarded by the router can also be stored locally. Edges in this network indicate neighbor relationships among the cache-routers, such that cache-router v can forward an unsatisfied request only to its neighbors. For the sake of readability, the term "caches" and "nodes" will be used interchangeably here to indicate these cache-router entities.

Let $F = \{f_1, ..., f_n\}$ be the set of unique items of content that can be requested in the network, termed here *files*. For a given state of the system we treat each node as a set or sequence[1] of stored elements at that state: $f_i \in v_j$ denotes that f_i is stored at the cache of v_j, $|v_j|$ is the size of the jth

---

[1]Random replacement policies ignore the position of a cached file, while LRU and FIFO are examples of replacement policies where the order is significant.

cache. We assume in this paper (for exposition purposes) that all files in the system are of identical size, and so the size of a cache is expressed in terms of the number of files it can store, and that all caches have identical size $|v|$. We discuss relaxing these assumptions in §VII-C.

Permanent Copies. In addition to the short-term storage provided by caches, we assume content is also stored permanently at one or more content *custodians* in the network, such as public content servers [21][17][22]. These custodians connect to the network at a specific location (i.e., a cache), and so we will denote them as a set $C \subseteq V$. Note that for each $v \in C$, the storage required for maintaining these permanent copies is not included in the specified cache size. We use $v \in cust(i)$ to denote that (a custodian connected to) node v has a permanent copy of f_i. For exposition purposes here we assume that for all $1 \le i \le n$, $|cust(i)| = 1$, though our results hold as long as $|cust(i)| \ge 1$. Since we assume $cust(i)$ is a singleton, we interchangeably refer to $cust(i)$ as a single node or a set containing one node.

Request Routing. Requests for content can arrive at a cache in two ways. Requests originate exogenously from users seeking content, who send requests into the network. Requests are then forwarded endogenously within the network until the request is satisfied. When a cache cannot satisfy a request, the request is forwarded along a *path* in the network in search of a copy. In this paper we assume there exists a static routing matrix $\mathcal{R} \in V^{m \times n}$, such that $\mathcal{R}(v_j, f_i)$ indicates the next-hop to forward unresolved requests for f_i at v_j, termed *cache misses*. To ensure all requests for f_i are eventually satisfied, we assume the path for every request ends at a node $v \in cust(i)$. A common example for a set of static paths is that of *shortest path* routing (used, for example, in [20]), in which a request for f_i is routed along the shortest path to the closest node in $cust(i)$.

Request Handling. A request for f_i arriving at node v_j is denoted (v_j, f_i). For all $f_i \in F$, $v_j \in V$, λ_{ij} is the exogenous rate of (v_j, f_i), where by "rate" we mean the *average* number of requests per time unit. When a request for f_i arrives at v_j, one of two things occurs:

- If $f_i \in v_j$, a cache *hit* occurs, and the file is forwarded back to the origin node where the request first entered the network. The file follows the reverse path that the request traversed.
- Otherwise, a cache *miss* occurs. If $v_j \in cust(i)$ then f_i is retrieved from this custodian; Otherwise, the request is forwarded to cache $\mathcal{R}(v_j, f_i)$.

Unless otherwise stated, when a file f_i is being downloaded and it passes through a node v_j with a full cache, one of the files in the cache will be *evicted* to make room for f_i (assuming $f_i \notin v_j$). A *replacement policy* at each cache determines which file is evicted.

We denote e_{ij} as the probability that $f_i \in v_j$. Also, let r_{ij} be the combined incoming rate of (v_j, f_i), and let m_{ij} be the rate of requests for f_i in the miss stream at node v_j. The rate

Notation	Meaning
v	A cache-router entity
$|v|$	The number of files a cache can store
f	A content entity (file)
m, n	The number of nodes and files, respectively
$cust(i) \subseteq V$	List of custodians for f_i
(v_j, f_i)	A request for f_i at cache v_j
e_{ij}	Probability that $f_i \in v_j$
$\mathcal{R}$	Request routing matrix
$a, b \in \Omega$	States in the Markov Chain system representation
$a[j]$	The content of v_j when system at state a

of (v_j, f_i) is then

$$r_{ij} = \lambda_{ij} + \sum_{h:\mathcal{R}(v_h, f_i)=v_j} m_{ih} \qquad (1)$$

B. Model Assumptions

In this work we adopt the following common modeling assumptions.

- **Arrival Process.** We use here a common model for the arrival process of exogenous requests - the Independent Reference Model (IRM) (e.g., [14], [17] and others), which states that the probability that the next *exogenous* request at a cache is independent of the earlier requests. Formally, let $X_k \in F$ be the kth file exogenously requested at some cache v. With IRM we have

$$Pr(X_k = f_i | X_1, \dots X_{k-1}) = Pr(X_k = f_i) \qquad (2)$$

- **Download Delay.** For exposition purposes, we assume that the time between a cache miss and the time when the file arrives at the cache is much smaller than the inter-request timescale and can therefore be ignored [17], [14], [19], [23]. Thus, once a cache miss occurs, the file is assumed to be instantaneously downloaded into the cache, as well at caches along the download path. We refer to this assumption as the *Zero Download Delay* (ZDD) property. In §VII-C we discuss how to extend our results here to non-ZDD systems.

C. Markov Model

We model a Cache Network as a discrete-time Markov chain with state space Ω_0. The system state s, $s = (s[1], s[2], \dots, s[m])$, is a concatenation of m vectors of size $|v_j|$ each, $1 \leq j \leq m$. The ith element in vector v_j corresponds to the content in the ith position of v_j. When all caches have the same size, the state space Ω_0 has cardinality $\left(\binom{n}{|v|} \cdot |v|! \right)^m$. When the order of the elements in the caches is irrelevant, states which differ only through such ordering are lumped together. In this case, when all caches have the same size, the state space $\Omega \subset \Omega_0$ has cardinality $\binom{n}{|v|}^m$.

A sample path τ_a of a CN is determined by its initial state, a, and a sequence of file requests and consequent evictions, σ and π, respectively, $\tau_a = (\sigma, \pi)$. Let $\sigma = (\sigma_k)_{1 \leq k \leq K}$ be a sequence of K file requests. Let $\pi = (\pi_k)_{1 \leq k \leq K}$ be a sequence of K file eviction sets. For each request σ_k, let

$\pi_k(h; \sigma) \in F \cup \{\star\}$ be the file evicted from cache h, $1 \leq h \leq m$, when request σ_k is served. $\pi_k(h; \sigma) = \star$ means that no file is evicted from h when request σ_k is served. Note that a single request can cause, along the file download path, changes at multiple caches. In addition, since we assume ZDD, we can ignore intermediate stages reflecting system states while content is being forwarded along its download path, as the probability that a request arrives during this time is modeled as being negligible.

Given initial state s, let s_k be the state resulting from the service of the k-th request in τ_s. Let $s_k(h)$ be the state of cache h at system state s_k. Let $\Gamma(\tau_s)$ be the sequence of states of the sample path τ_s, $\Gamma(\tau_s) = (s_1, \dots, s_K)$, where $s_1 = s$. $\varphi(\tau_s)$ is the last state of $\Gamma(\tau_s)$, $\varphi(\tau_s) = s_K$.

Let $\mathbf{A} = (a_{c,d})_{1 \leq c,d \leq m}$ be the adjacency matrix of a CN. $\mathbf{A}$ is a binary matrix, where $a_{c,d} = 1$ if it is possible to reach state d from state c through one transition,

$$a_{c,d} = \begin{cases} 1, & \exists \tau_c = (\sigma, \pi) : d = \varphi(\tau_c) \wedge |\sigma| = 1 \\ 0, & \text{otherwise} \end{cases} \qquad (3)$$

State d can be reached from state c if there is a sample path $\tau_c = (\sigma, \pi)$ such that $d = \varphi(\tau_c)$. Let $a_{c,d}^{(n)}$ be an element of $\mathbf{A}^n$, $1 \leq c, d \leq m$. State d can be reached from state c if there exists an integer n such that $a_{c,d}^{(n)} = 1$. We conclude with some terminology used throughout the paper:

Definition 1: A *transient state* of a CN is a state that can be left and never reached again.

Definition 2: A *recurrent state* of a CN is a state that is not transient.

Definition 3: An *ergodic set* of a CN is a set in which every state can be reached from every other state, and which cannot be left once it is entered.

Definition 4: An *ergodic CN* is a CN that comprises a single ergodic set.

Note that the last definition slightly differs from the traditional definition of an ergodic Markov chain, whose states form a single ergodic set [24].

III. SENSITIVITY TO THE INITIAL STATE: EXAMPLES

To motivate the need for considering the initial state of the CN, we present here two scenarios in which the initial conditions of the CN determine its steady-state behavior, and consequently the performance of the CN system.

A. Example 1

In our first example we consider the topology in Figure 1. Let A, B be two disjoint sets of files, such that $|A| = |B| = |v|$, and assume LRU replacement is used at both caches. User 1 requests only files of type A, and user 2 only files of type B. Now, we consider two initial states: (I) when both caches are empty, and (II) when v_1 (v_2) contains exactly the set of files A (B). For scenario II the system will remain in the initial state indefinitely, with each cache storing only files from a single set, and will experience no cache misses. For scenario I, on the other hand, cache misses will occur indefinitely, and both caches will store over time files from both sets.

While this example is outwardly simple, it contains several interesting lessons, beyond the impact on performance. First, it is important to note that, with this user request pattern, the state space is disjoint: the initial state described in II *cannot be reached* from any other state. Second, slight changes in the request patterns of users can lead to drastically different cache behaviors. For example, if each user requests a set of files C_1, C_2, s.t. $|C_i| = |v|$ and $C_i \cap A \neq \emptyset, C_i \cap B \neq \emptyset$ for users $i = 1, 2$, the system would eventually converge to a single state, $v_i = C_i$, mirroring the user demand. Thus, we can see dependencies form in the network in such a way that, at times, small changes in user demand can have a very significant impact on cache behavior.
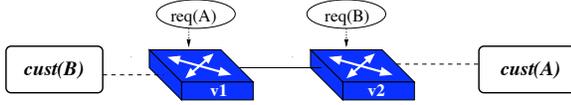

Fig. 1. Example scenario in which the solution of the MC is dependent on initial state. Cube-shaped routers indicate cache-routers, rectangles denote custodians (with the stored FIDs indicated within each), and elipses indicate user demand. Solid lines indicate an edge in E and dotted lines indicate a cache-custodian or cache-user association.

B. Example 2

Another example is a network comprised of caches using the FIFO replacement policy and $n = |v| + 1$ files in the network, with a non-zero request rate for each of these files at each node. A cache is *in isolation* if it has one or more directly attached custodians, which can provide any requested file to the cache. In §III-B1 we demonstrate that though the caches considered in this section are non-ergodic in isolation, their performance is independent of the initial state. Next, in §III-B2, we show that once the caches are interconnected (see Figure 3), the initial state is crucial for system performance.

1) A single FIFO cache in isolation: Here we show that a single FIFO cache with the given $n = |v| + 1$ ratio is non-ergodic. The set of states for which the cache is full can be partitioned into $(n-1)!$ disjoint sets of states, such that a state is reachable from another only if they are within the same set. Each of these sets of states corresponds to a cyclical ordering of the files, indicating the order in which they are evicted - an order which repeats itself indefinitely. Thus, the initial state (or, if we start with an empty cache, the first $|v|$ unique files requested), determines the steady-state of the cache. Despite this fact the probability that $f_i \in v$ is *independent* of the initial state. This can be determined from the balance equations,

$$(1 - e_i)\lambda_i = (1 - e_j)\lambda_j, \qquad 1 \leq i < j \leq n \quad (4)$$

$$\sum_{k=1}^{n} e_k = 2, \quad 0 \leq e_j \leq 1, \qquad 1 \leq j \leq n \quad (5)$$

where e_i is the probability $f_i \in v$. The system of equations (4)-(5) admits a single solution, independent of the initial state,

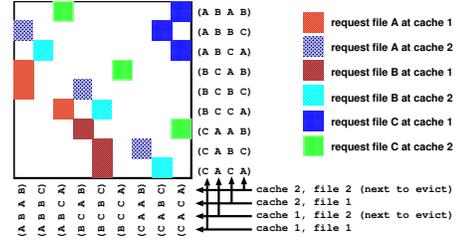$$e_i = 1 - \left(\lambda_i \sum_{j=1}^{n} \frac{1}{\lambda_j} \right)^{-1}, \qquad 1 \leq i \leq n \quad (6)$$

Fig. 2. Transition matrix of Example 2 (diagonal elements not shown). The system state is (w, x, y, z) where w and x (resp., y and z) are the two files at cache 1 (resp., 2). $A = f_1$, $B = f_2$, $C = f_3$ for initial state (f_1, f_2, f_1, f_2). $A = f_1$, $B = f_3$, $C = f_2$ for initial state is (f_1, f_3, f_1, f_3).

Since the arrival process is IRM, the occupancy probability is also the hit probability [18].

2) Dependencies in networks: Next, we show that the interconnection of the two isolated FIFO caches described in the previous section yields a CN in which the initial state impacts steady state performance. The caches are interconnected as shown in Figure 3. The topology consists of two caches, v_1 and v_2, arranged in a line, and a custodian connected to v_2 that can provide any of the three requested files. Each cache has size 2, $|v| = 2$. Upon a cache miss, the request is forwarded in the direction of the custodian. Figure 2 shows the transition probability matrix.

TABLE II
EXAMPLE OF THE IMPACT OF INITIAL STATE ON SYSTEM SOLUTION FOR THE TOPOLOGY IN FIG. 3 AND TRANSITION MATRIX SHOWN IN FIG. 2.

Initial State	e_{12}	e_{22}	e_{32}
(f_1, f_2)	0.46651	0.63134	0.90214
(f_1, f_3)	0.33054	0.76861	0.90083

Fig. 3. Example in which the solution of the MC depends on initial state. Cube-shaped routers indicate cache-routers, while rectangles with rounded edges denote custodians, that permanently store the indicated files. Solid lines indicate an edge in E and dotted lines a cache-custodian association.

To illustrate the impact of the initial state on the steady state solution, we initialize both caches at the same state, and consider two different initial conditions, $v_1 = v_2 = (f_1, f_2)$ and $v_1 = v_2 = (f_1, f_3)$. Let $\lambda_{1,1} = 0.35$, $\lambda_{2,1} = 0.55$, $\lambda_{3,1} = 0.1$, $\lambda_{1,2} = 0.05$, $\lambda_{2,2} = 0.15$, $\lambda_{3,2} = 0.8$. Table II shows the steady state file occupancy probabilities at cache 2, as a function of the initial state. It is clear from these results that, for this system, the initial state has substantial impact on the overall steady state performance.

Let us consider what is happening here. In the case of a single cache, the system is non-ergodic but, due to a symmetry among the states in the Markov model, the performance of this cache is unaffected by the initial state or requests. Once the caches are networked, however, this symmetry no longer applies, and a different steady-state distribution of files is obtained, depending on the initial conditions. Once again we see how it is the networking of caches that introduces unexpected behaviors.

IV. CONDITIONS FOR ERGODICITY: TOPOLOGY AND ADMISSION CONTROL

In light of the examples presented in the previous section, we present here several theorems regarding the ergodicity (or lack thereof) of a CN. Each theorem presents an independently-sufficient condition for ergodicity. We begin with several definitions.

Definition 5: The topology of a cache network is a *hierarchy* or *tree* if it has a single custodian and requests are forwarded along the shortest path to this custodian. For such structures, level i in the hierarchy includes nodes that are i hops away from the custodian, and $level(v_j) = i$ iff v_j is at the ith level.

Definition 6: An exogenous request stream for files at v_j is said to be *positive* iff $\forall f_i \in F$, $\lambda_{ij} > 0$. If this condition holds for all $v \in V$, we say the exogenous request stream at the (cache) network is positive.

Recall that a cache in isolation is connected to a custodian that can provide any requested file to the cache (see §III-B).

Definition 7: A CN is said to be *individually ergodic* if its components are ergodic in isolation. This means that, for each cache $v \in V$, when v functions as a cache in isolation, v is ergodic, given the exogenous request stream is positive.

The example in §III-A used a non-positive request stream, while the example in §III-B uses a cache hierarchy that is not individually ergodic. Note that, though we are not aware of such a system, it is conceivable that due to dependencies between the states of neighboring caches, an individually ergodic system will be non-ergodic as a whole. We discuss this possibility in §VIII. We now state our first two theorems.

Theorem 1: An individually-ergodic CN with positive exogenous request streams is ergodic if it is a hierarchy.

Theorem 2: Consider an individually-ergodic CN where v_j caches file f_i (if and when f_i passes through v_j) with probability $0 < \theta_{ij} < 1$ for all $1 \leq i \leq n$ and $1 \leq j \leq m$. Then this system is ergodic when subject to positive exogenous request streams.

These two theorems are proven using the same general approach: We begin with selecting a pair of states $a, b \in \Omega$ and demonstrate that there exists a sample path between them. In other words, we show that there is a series of requests and evictions that change the system state from a to b. Since the system is of finite size, this implies ergodicity of the entire system.

The proof for Theorem 1 relies on the following observation regarding cache trees:

Lemma 1: When ZDD is assumed, caches at the ith level of a hierarchy are unaffected by the state or request stream experienced at caches of level $i' \leq i$.

Proof: In a hierarchy, caches at the same level do not communicate directly, as all requests are sent upstream and all content is forwarded downstream. Thus, the state at a cache at the same level as v_j does not directly impact v_j. Thus, we need only prove the claim here for $i' < i$.

The state at a node is determined by the flow of content through it. This flow is affected by two factors: the requests that arrive at the cache, and the content at the cache when these arrive (which determines cache misses). In a hierarchy, requests do not get forwarded downstream from caches at upper levels, and so the request stream seen at level i is not affected by the state of nodes higher up. The state at a cache can, of course, be affected by the time it takes to download content. However, since we assume ZDD, the time it takes for a cache miss to be satisfied is assumed negligible, and so the state a v_j is not impacted by cache misses or hits at upper levels. With a custodian for each piece of content, there are no unsatisfied requests, so the claim is proven. ∎

Proof of Theorem 1: To transition from a to b, we iterate over all caches, starting from the lowest level l and working our way up (within each level the order of caches is arbitrary), and generating a sequence of exogenous requests at each cache that will modify it from state $a[j]$ to $b[j]$. Using Lemma 1, we argue this process will result in a sample path from a to b.

Formally, we index the nodes starting at the root, s.t. if $level(v_h) > level(v_j)$ then $h > j$, and iterate over the caches in descending order of cache indices. At each node v_j, since it is individually ergodic and the exogenous requests are positive, there exists a series of exogenous requests (and evictions) that will modify the state at this node from $a[j]$ to $b[j]$. By using exogenous requests, we do not rely on the state of caches at lower levels, and from Lemma 1 we also know and that this transition has no impact on the state of caches with a larger index. Thus, once we get to v_1 we know by induction that the system is in state b. ∎

Proof of Theorem 2: We assume that for each cache v_j $\theta_{ij} < 1$ for all files. Thus, for any finite sequence of files σ that pass through this node and any subsequence σ' of σ, there is a positive probability that only the files in σ' will be admitted to the cache for storage. Thus, we iterate over the caches in arbitrary order, and at each cache let σ_j be a sequence of requests originating from v_j, s.t. applying these requests yields $a[j] \rightsquigarrow b[j]$ - since the caches are individually ergodic, such a sequence exists. In addition, there is a non-zero probability that only v_j will store the files requested in σ_j. This series of events will therefore result in the path $a \rightsquigarrow b$. ∎

V. CONDITIONS FOR ERGODICITY: REPLACEMENT POLICY

We now proceed to present a key contribution of this paper - a theorem regarding the impact of a replacement policy on the ergodicity of the system. We shall begin in §V-A with a narrow formulation considering only Random replacement, and then in §V-B expand this result to a broad class of replacement policies.

A. Theorem for Random Replacement

In this section we consider the Random replacement policy. A CN in which all caches use the Random replacement policy, with positive exogenous request streams, is individually ergodic. In general, whether individually ergodic CNs are ergodic is an open question. Nevertheless, when caches use

the Random replacement policy the answer to the question is *yes*, as stated in the following theorem.

Theorem 3: A CN that uses Random replacement is ergodic when subject to positive exogenous request streams.

Let $a, b \in \Omega$ be two *recurrent* states. We prove our claim by showing there exists a state c that is reachable from both a and b. Since we assume a and b are recurrent, there must exist a reverse path from c to each of them, and so a and b communicate with eachother. This concludes our proof, since it shows there is a single ergodic set of states in this system.

Our proof will proceed by considering two CNs that are identical in all aspects (topology, routing, cache size and replacement policy, custodian location) except in their initial state - CN_a begins in state a and CN_b in state b. Given their states, we generate a sequence of exogenous requests $\boldsymbol{\sigma}$, which will arrive at both CNs. To accommodate the differences in the initial state, we will also design two sequences of evictions $\boldsymbol{\pi}_a$ and $\boldsymbol{\pi}_b$ to match $\boldsymbol{\sigma}$, and demonstrate that the sample path in both networks leads to the same state. In fact, we will design these paths so that both networks are monotonically becoming more similar to one another. To quantify this, we use the following definition: Let $\gamma_{a,b}(j) = |a[j] \cap b[j]|$ be the agreement index of two networks at v_h. We say that two caches *agree* iff $\gamma_{a,b}(j) = |v|$. As we will demonstrate for the sequence we construct, for all $1 \leq k < h \leq K$ and all $1 \leq j \leq m$, $\gamma_{a_k,b_k}(j) \leq \gamma_{a_h,b_h}(j)$, and after $\boldsymbol{\sigma}$ is served $1 \leq j \leq m$, $\gamma_{a_K,b_K}(j) = |v|$. In what follows, we formalize these intuitions by showing how to construct the sequences of file requests and evictions.

Requests. Consider two CNs, CN_a and CN_b, which differ only in their initial states, a and b, respectively. Algorithm 1 describes how to construct the sequence of requests $\boldsymbol{\sigma}$ to be applied in each of these networks. Our approach will be to iterate over all the caches (line 2), and for each cache ensure that its state is the same for both CNs. Specifically, for cache v_j, Δ_j is the set of files in the cache in CN_b but not in CN_a (line 3). Then, for each file f_i we generate requests for f_i at each cache along the path from $cust(i)$ to v_j in both CNs according to $\mathcal{R}$. We do so by injecting an exogenous request at each node along this path (lines 5-11).

Evictions. Next, our goal is to determine the evictions that will take place in both networks. Assume during execution of $\sigma_k \in \boldsymbol{\sigma}$ file f_i arrives at v_h. What to evict will depend on the state of each of the two networks at this stage:

- $f_i \in a_k[h] \cap b_k[h]$ - no evictions are needed, since in both networks the file is already cached at v_h. Here, $a_k[h]$ and $b_k[h]$ refer to the set of files at node h after processing request σ_k at CN_a and CN_b, respectively.
- $f_i \notin a_k[h] \cup b_k[h]$ - in both networks, the cache does not have the file. We cache f_i in each, and evict some file from each.
  - If $a_k[h] \cap b_k[h] = \emptyset$, select a random file from each to evict. This is called a *random two-sided eviction*.
  - Otherwise, select a file $f \in a_k[h] \cap b_k[h]$ to evict. This is called an *identical two-sided eviction*

Algorithm 1 SigmaConstruct(a, b).

Input: $a, b \in \Omega$ recurrent states in the Markov chain representing the CN

1: $\boldsymbol{\sigma} \leftarrow ()$ // Empty sequence
2: **for** $j = 1 \to m$ **do**
3:     $\Delta_j \leftarrow b[j] \setminus a[j]$
4:     **for** $f_i \in \Delta_j$ **do**
5:         $\boldsymbol{\sigma}' \leftarrow ()$
6:         current = j
7:         **while** $v_{current} \neq cust(i)$ **do**
8:             $\boldsymbol{\sigma}' \leftarrow (v_{current}, f_i)|\boldsymbol{\sigma}'$ // "|" means concatenation
9:             current = $\mathcal{R}(v_j, f_i)$
10:         **end while**
11:         $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}|\boldsymbol{\sigma}'$
12:     **end for**
13: **end for**
14: return $\boldsymbol{\sigma}$

- Otherwise, w.l.o.g. $f_i \in b_k[h] \setminus a_k[h]$. Then we change nothing at $b_k[h]$, and evict from $a_k[h]$ some file $f' \in a_k[h] \setminus b_k[h]$. We call this a *one-sided eviction*.

Lemma 2: Following the eviction rules above, $b_k[j] \setminus a_k[j] \subseteq b_h[j] \setminus a_h[j]$ for all $h < k$.

Proof: Let us consider a file f_i, requested at σ_l, $h < l \leq k$, causing an eviction at node v_j. A one-sided eviction increases agreement of caches, since a non-matching file was evicted to make room for a matching file. A random two-sided eviction increases agreement, since beforehand the caches had no files in common, and now they share f_i. Finally, with an identical two-sided eviction, file $f_q \in b_h[j] \cap a_h[j]$ was evicted from both caches to make room for f_i, which does not decrease the cache agreement. ∎

Lemma 3: After the files in Δ_j were requested in $\boldsymbol{\sigma}$, both networks agree on v_j.

Proof: Assume we Δ_{j-1} ended with request σ_k. From Lemma 2 we know that $b_k[j] \setminus a_k[j] \subseteq \Delta_j$. Every file that is in $b_k[j] \setminus a_k[j]$ will therefore cause a one-sided eviction, increasing agreement by 1, and every other file does not decrease agreement. Thus at the end of Δ_j the networks agree at v_j. ∎

We use this construction to prove our Theorem.

Proof of Theorem 3: We prove Theorem 3 using an inductive argument. From Lemma 3 we know that after requesting Δ_j both networks agree on the state of v_j. Furthermore, we know from Lemma 2 that no download can negatively impact cache agreement at v_j in the two CNs in the future, so once caches agree this remains so. Thus, after requesting all the mismatched files at all caches, the networks agree. ∎

B. From Random Replacement to non-protective policies

A review of our proof for Theorem 3 reveals that the only place in which it relied on using Random replacement was in assuming that, given that an eviction is taking place, there is a transition in the Markov chain for evicting each of the

files at that node. This allowed us more freedom in defining a sample path between two designated states. Using this insight, we consider the following class of policies:

Definition 8: A replacement policy for an isolated cache is said to be *non-protective* if for any file $f \in v$ there is a positive probability that f will be the next file to be evicted (if another eviction takes place). A replacement policy for which this property does not hold will be termed *protective.*

Note that a cache using a non-protective replacement policy is individually ergodic. While with Random replacement the next eviction could be any file, this definition is broader. It covers policies in which requests can change the later order of evictions at a cache without changing its contents. LRU is an example of such a policy, since a request for $f_i \in v_j$ that arrives at v_j can change the later eviction order even though the files in the cache do not change as a result of the arrival for request f_i. The same holds for other policies such as LRU-K and LFU. FIFO, on the other hand, is a protective policy when $|v| > 1$. In this section, we prove the following extension of Theorem 3:

Theorem 4: A CN with positive exogenous request streams is ergodic if the replacement policy used in each cache is non-protective.

Note that this theorem allows for *heterogeneous* systems where different caches could use a different (non-protective) replacement policy. Our approach will be to demonstrate that ergodicity of Random replacement can be used to prove the ergodicity of all systems where the caches are non-protective. At a high level, given a sample path of the Markov chain of a CN with Random replacement caches, we add exogenous requests at each cache to create a sample path in which the ordering of evictions that take place will be the same as with the targeted non-protective replacement policy. For exposition purposes, our examples will use LRU replacement, though the proof applies to all non-protective policies.

Unlike Random replacement, other policies use an (explicit or implicit) ordering of the files in the cache, from which the eviction order can be determined. With LRU, for example, items are ordered according to last reference. We begin here by "lumping" together, in the LRU model, all the states that differ only in the internal ordering of content, and mapping these to the state in the Random model. Figure 4 depicts an example of such a mapping for a 2-node cache, while Figure 5 does the same for FIFO.

These examples demonstrate that changes in eviction order that do not change the content of any cache are possible within LRU networks but not within FIFO networks. For the former, the closure of each such "lump" of states is a clique, with every two states communicating with one another only via other states with the same content. For the latter, even for an isolated cache a change in order can only be achieved by changing the content of the cache, and in networked scenarios the situation is complicated by the fact that during content download other caches might be impacted as well. This makes determining ergodicity for FIFO and other protective policies more challenging, since a state-request pair fully or partially

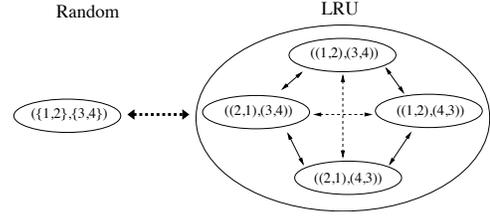determines the files to be evicted, limiting us in finding a path between two recurrent states.

Fig. 4. RND-to-LRU state mapping and edge contractions example. Edges indicate transitions in the Markov model. As can be seen here, the closure of the indicated transitions results in a clique (broken edges mark the added connectivity), so it is possible to move from any state to any other without influencing the set of files stored in any cache.
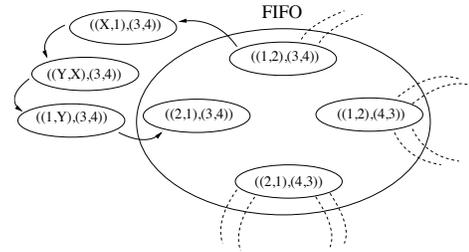

Fig. 5. An example for the situation with FIFO replacement. $X, Y \in F \setminus \{1, 2\}$. As can be seen here, with FIFO there are no edges between states with the same content in all the caches, and all paths between such states require changing the content of some caches. In fact, there is no way to change the order of eviction in a cache with FIFO.

Proof of Theorem 4: Let M_{alg} be the Markov chain of a CN using a replacement algorithm *alg*. Furthermore, let M_{alg}^* be the same graph, but after contracting all nodes representing identical cache contents. After the contraction, edges between states with the same content are eliminated, and all edges with other states are attached to the contraction node. We first demonstrate that $M_{alg}^* = M_{rnd}$ if *alg* is non-protective. Next, we show that when *alg* is non-protective, if M_{alg}^* is ergodic, so is M_{alg}. Since we know M_{rnd} is ergodic, the theorem is proven.

As discussed above, each node in M_{rnd} is mapped to the node in M_{alg}^* representing all nodes in M_{alg} where caches have the same content. Next, consider two states in M_{rnd} and their mapped counterparts, denoted $s_{rnd}, s'_{rnd}, s_{alg*}, s'_{alg*}$. We want to prove there is an edge (s_{rnd}, s'_{rnd}) iff there exists an edge (s_{alg*}, s'_{alg*}).

- First, we note that in s_{rnd} and s_{alg*} each cache holds the same content. Thus, a request will traverse the same caches in both networks regardless of replacement algorithm, and be stored at the same caches.
- Second, we show that the same evictions can take place in both. Consider a specific cache, v_j. With Random replacement all content in v_j is up for eviction. Similarly, since *alg* is non-protective, for every $f_i \in v_j$ there is a state in the set of contracted states that evicts this file. Thus, an edge reflecting this eviction will exist.

Thus, the equivalent of these graphs is proven, and one is ergodic iff the other is ergodic. We now demonstrate that since M_{alg}^* is ergodic, M_{alg} is as well. Since with non-protective systems there is a path between any two states that share the same cache contents, without moving outside this set of states, this claim is true. For each request σ_k and eviction set π_k for random replacement, we can augment the sample path with requests between σ_{k-1} and σ_k that cause no eviction, but arrange the content in the caches such that when σ_k is served the same π_k are evicted. This is possible due to the non-protective property of the replacement policy. Thus, since all states within the contracted states communicate, and all recurrent states in M_{alg}^* communicate (following ergodicity), we conclude that M_{alg} is ergodic. ∎

VI. RELATED WORK

As caches are part of many computing and communication systems, it is no surprise that the the literature on caches, and cache networks in particular, is vast. In this section we survey some common models developed in the literature for these systems. Despite the large volume of work on this topic, to the best of our knowledge this work is the first to study the ergodicity of cache networks.

A. Results for stand-alone and networked caches

Work on isolated caches abounds, and surveying it is beyond the scope of this section. A partial survey of common replacement policies can be found in [25], [26]. In general, it is accepted that modeling even a stand-alone cache using classic replacement policies (e.g., LRU, FIFO) in a precise manner is intractable, due to state explosion as the cache size and number of files grow[13]. As a result, fast approximations have been proposed for these caches [14].

Most of the work we are aware of has focused on the steady-state of these systems, but there has also been interest in the behavior during the transient stage as a way to model behavior under traffic surges [27]. In this work, the authors discuss the warmup phase of LRU and how LRU deals with warming up when starting from a non-empty initial stage. Our interest here differs in that we consider cache networks, and we focus on the resulting steady state of the system, which to the best of our knowledge has not been discussed before.

Moving on to networked caches, there has been substantial work regarding cache *hierarchies* or *trees* [19], [10], [20], [28], [11], [26], [29], [30]. These systems are characterized by the existence of a single content custodian for all content (e.g., slow memory for file-systems, the Internet for web proxy caches) and shortest-path routing used for request routing. The combination of these two factors causes requests to flow only upstream, from caches towards the custodian, and content to flow only in the opposite direction. This special structure allows for analysis to be done from the bottom up, as is done in the aforementioned papers, and we make use of this structure in Theorem 1. In our work we also tackle networks with arbitrary topologies, where content and requests can flow in both directions on network links.

Most work on cache networks (and similar systems - see below) has generated analytical bounds only for certain classes of topologies and specific replacement policies. To the best of our knowledge, the main theorem in this paper is the first theoretical claim that applies for a wide range of replacement policies, *mixtures* of such policies, and for *arbitrary* network topologies.

B. The P2P connection

A large body of work that bears much resemblance to these systems is that of P2P networks, especially *hybrid unstructured P2P networks* [21], [17], [23], [22]. In these systems, peers form an overlay network of arbitrary topology search for content among peers in this network and then download it. The system is termed "hybrid" because it assumes there is always a publisher entity available in the system, to which clients can turn to if content is not available in the P2P "swarm". Thus, publishers and peers have similar roles to custodians and caches, suggesting that results might be transferable from one field to the other. There are, however, some important differences. One such difference is that with P2P systems the topology is relevant only for content search while in CNs content download also takes place with this topology, and content is stored at nodes along the download path. In this sense cache networks are a generalization of these systems, in which the download path is taken into account.

VII. DISCUSSION

In this section we discuss some of our model assumptions. All the components of the model we adopt here are used elsewhere in the literature. These include ZDD [17], [23], [19], [10], IRM requests [14], [21], [19], [10], [17], [23], identical exogenous access patterns at all users [21], [23], [28] and storing content at nodes that did not request it (i.e., $\theta_{ij} > 0$) [22]. In this section we discuss in more detail some of our main modeling assumptions - IRM exogenous arrivals, cache coordination and homogeneous cache policies.

A. IRM Exogenous Request Streams

The model we use here for exogenous request traffic is, as mentioned in the previous section, the Independent Reference Model (IRM). However, there are alternative models for request patterns at single caches. Panagakis et. al. [31] present approximate analysis for streams that have short term correlations for requests. Since in cache network we see the opposite effect, with content requested recently *less* likely to be requested next, this approach is less useful here without modifications. An approach that deviates sharply from IRM is that of Stack Depth Distribution (SDD) [32]. With this model, the stream of requests is characterized as a distribution $\vec{h} = (h_i)_{i=1}^{\infty}$ over the cache slots in a cache of infinite capacity, where h_i is the probability that the next cache hit will be at slot i. In this model, all information regarding the individual files being requested is ignored or unavailable, and so is not used here.

B. System Architecture - Cache Coordination

When considering cache interaction, different approaches have been suggested regarding cache coordination. Some have proposed systems where caches explicitly coordinate what to store and where [33][34], while at the other end of the spectrum some have considered systems where caches are oblivious to the existence of other caches [19][6][18]. The question of ergodicity is most pressing for this second category of systems, and so we chose it for our model in this paper, though the tools presented here could be used for some coordinating systems as well.

C. Relaxing the Model

Throughout the paper we assumed ZDD and that files and caches were all of constant size. While these make the exposition simpler, the proofs we present here apply equally when these restrictions are removed. Recall that our method of proof was to demonstrate that there exists a sample path of the Markov chain that leads from one state to another. Even when we relax the ZDD assumption, it is still *possible* that every request was satisfied before the next one was generated. Thus, the same path we constructed in each of the proofs will exist in this finer-granularity environment as well.

Regarding cache sizes, the proofs apply as-is to variable cache sizes, by changing each $|v|$ to $|v_j|$ for the jth cache. Similarly, allowing for files to have variable size does not interfere with our proof concept, as long as (when needed) we allow evicting a set of smaller files to make room for a single large file. The proofs for Theorems 1 and 2 do not rely on file sizes. For Theorem 4, the request sequence described in the proof is constructed in the same manner, and when evictions take place, each eviction type can be shown to maintain or improve cache agreement.

VIII. SUMMARY AND FUTURE WORK

In this paper we demonstrated the importance of rigorous analysis of cache networks, and proved several theorems regarding the ergodicity of these systems. While solving a Markov model of a cache network is intractable for any internet-scale system, we have shown here that one can still use these models to make *structural* arguments that may lead to interesting insights. We hope to further explore the potential of such arguments in the future.

The examples presented in §III did not include cases with positive request streams *and* individually ergodic systems. We pose as an open question if there exists such a system that is not ergodic. Theoretically, such a system could exist, with non-ergodic behavior on a system-wide level caused by dependencies among caches, formed by the file download paths. At this time, however, we are unaware of such a system. We hope that future investigations will shed light on this question.

REFERENCES

[1] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "Voccn: voice-over content-centric networks," in *ReArch '09*. New York, NY, USA: ACM, 2009, pp. 1–6.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT*, 2009.

[3] K. Pentikousis and T. Rautio, "A multiaccess network of information," in *IEEE WoWMoM*, june 2010, pp. 1 –9.

[4] K. Katsaros, G. Xylomenos, and G. Polyzos, "A hybrid overlay multicast and caching scheme for information-centric networking," in *IEEE Global Internet Symposium*, march 2010, pp. 1–6.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A scalable content-addressable network," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, pp. 161–172, 2001.

[6] E. Rosensweig and J. Kurose, "Breadcrumbs: efficient, best-effort content location in cache networks," in *IEEE INFOCOM*, 2009.

[7] D. Trossen, M. Sarela, and K. Sollins, "Arguments for an information-centric internetworking architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 40, pp. 26–33, April 2010.

[8] G. Peng, "Cdn: Content distribution network," *ArXiv cs/0411069*, 2004.

[9] L. Qiu, V. Padmanabhan, and G. Voelker, "On the placement of web server replicas," in *INFOCOM 2001*, vol. 3, 2001, pp. 1587 –1596 vol.3.

[10] N. Laoutaris, H. Che, and I. Stavrakakis, "The lcd interconnection of lru caches and its analysis," *Performance Evaluation*, vol. 63, 2006.

[11] M. Busari and C. L. Williamson, "Simulation evaluation of a heterogeneous web proxy caching hierarchy," in *MASCOTS*. IEEE, 2001.

[12] L. Breslau, P. Cue, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *IEEE INFOCOM*, 1999, pp. 126–134.

[13] W. F. King, "Analysis of paging algorithms," in *IFIP Congress*, 1971.

[14] A. Dan and D. F. Towsley, "An approximate analysis of the lru and fifo buffer replacement schemes," in *SIGMETRICS*, 1990, pp. 143–152.

[15] R. Fonseca, V. Almeida, M. Crovella, and B. Abrahao, "On the intrinsic locality properties of web reference streams," in *IIEE INFOCOM*, 2003.

[16] C. Williamson, "On filter effects in web caching hierarchies," *ACM Trans. Internet Technol.*, vol. 2, pp. 47–77, February 2002.

[17] S. Ioannidis and P. Marbach, "On the design of hybrid peer-to-peer systems," in *SIGMETRICS*, 2008.

[18] E. Rosensweig, J. Kurose, and D. Towsley, "Approximate models for general cache networks," in *IEEE INFOCOM*, march 2010, pp. 1 –9.

[19] H. Che, Z. Wang, and Y. Tung, "Analysis and design of hierarchical web caching systems," in *IEEE INFOCOM*, 2001, pp. 1416–1424.

[20] G. Carogli, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," France Telec., Tech. Rep., 2011.

[21] S. Tewari and L. Kleinrock, "Proportional replication in peer-to-peer networks," in *IEEE INFOCOM*, april 2006, pp. 1 –12.

[22] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks," in *ICS*. New York, NY: ACM, 2002, pp. 84–95.

[23] S. Ioannidis and P. Marbach, "Absence of evidence as evidence of absence: A simple mechanism for scalable p2p search," in *IEEE INFOCOM*, 2009.

[24] J. Kemeny and J. Snell, *Finite Markov Chains*. Springer, 1976.

[25] I. Ari, "Design and management of globally distributed network caches," Ph.D. dissertation, UC Santa Cruz, 2004.

[26] Y. Zhou, Z. Chen, and K. Li, "Second-level buffer cache management," *Parallel and Distributed Systems, IEEE Trans. on*, vol. 15, no. 6, 2004.

[27] A. Bhide, A. Dan, and D. Dias, "A simple analysis of the lru buffer policy and its relationship to buffer warm-up transient," in *Ninth International Conference on Data Engineering*, apr 1993, pp. 125 –133.

[28] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *IEEE INFOCOM*, 2010, pp. 1 –9.

[29] R. Gupta, S. Tokekar, and D. Mishra, "A paramount pair of cache replacement algorithms on l1 and l2 using multiple databases with security," in *ICETET*, dec. 2009, pp. 346 –351.

[30] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modeling and evaluation of ccn-caching trees," in *IFIP Networking*, 2011.

[31] A. Panagakis, A. Vaios, and I. Stavrakakis, "Approximate analysis of lru in the case of short term correlations," *Computer Networks*, vol. 52, no. 6, pp. 1142–1152, 2008.

[32] H. Levy and R. J. T. Morris, "Exact analysis of bernoulli superposition of streams into a least recently used cache," *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 682–688, 1995.

[33] X. Tang and S. T. Chanson, "Coordinated en-route web caching," *IEEE Transactions on Computers*, vol. 51, no. 6, pp. 595 – 607, 2002.

[34] M. R. Korupolu and M. Dahlin, "Coordinated placement and replacement for large-scale distributed caches," *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 6, pp. 1317–1329, 2002.