BISON
IST-2001-38923

Biology-Inspired techniques for Self Organization in dynamic Networks

Models for Advanced Services in AHN, P2P Networks

Deliverable Number:	D08
Delivery Date:	January 2004
Classification:	Public
Contact Authors:	Andreas Deutsch, Niloy Ganguly, Geoffrey Canright, Márk Jelasity, Kenth Engø-Monsen
Document Version:	Final (January 30, 2004)
Contract Start Date:	1 January 2003
Duration:	36 months
Project Coordinator:	Università di Bologna (Italy)
Partners:	Telenor Communication AS (Norway), Technische Universität Dresden (Germany), IDSIA (Switzerland), Santa Fe Institute (USA)

Project funded by the European Commission under the Information Society Technologies Programme of the 5th Framework (1998-2002)

Abstract

This document describes the characteristics of the models that have been developed so far in BISON in relationship to the management of advanced functions in dynamic networks. In order to cover a wide spectrum of possible dynamic networks scenarios, in this phase of the project we have investigated a number of different models. The models include various load balancing schemes and schemes to efficiently distribute contents across the network. Moreover, a modular approach through which advanced functions are realized by combining various basic functions is also proposed. The schemes are inspired by different biological concepts like epidemics, ants, chemotaxis, immune system etc. In each of the models, fairness criteria have been defined to compare the model with state of the art technology. Initial experimental results are also provided along with each model. The preliminary evaluation results that have been studied so far appear promising according to the BISON's objectives.

Contents

1 Introduction

This deliverable deals with advanced functions. The "advanced" functions to be studied by BISON differ from the "basic" functions (discussed in D05) in their complexity. They involve the integrated performance of multiple, more basic functions. Hence advanced functions can be generally 'decomposed' in terms of more basic functions. We consider three advanced functions in our discussion. The functions are namely distributed processing, distributed storage and distributed content sharing. In this deliverable, we present a brief description of each of the functions. The brief description is also contained in D01, however, we present it here also for the sake of completeness of the deliverable. The state of the art is described next which brings forward the necessity of *biologically inspired algorithms*. Three initial works have been undertaken till now. The basic schemes and preliminary results of these works are next discussed. Finally, we summarize the deliverable in the 'Outlook' section.

The advanced functions have historically been considered almost solely in the context of overlay networks built over the Internet. The reason for this is that these functions involve the sharing, and hence more effective utilization, of various types of capacity (processing, storage, bandwidth) present at the participating nodes. The collective capacity (processing, storage and bandwidth) of all the PCs connected to the Internet is enormous. In contrast, ad-hoc networks tend to be small (and confined to the research stage)—and the typical mobile terminal participating in an ad-hoc net has (again, historically) relatively limited capacities of all three types. Hence, we believe in the near future, except for some partial implementation of distributed content, none of the other functions will be important in ad-hoc networks.

Distributed processing

Here the capacity to be shared is processing power. Distributed processing is best suited for processing tasks that can be broken into nearly-independent pieces so that there is need for little coordination between the processes going on at different nodes. In other words, the emphasis is on the utilization of highly distributed processing power, with the attendant practical constraint that the resulting parallel processing must seek to minimize the interprocessor communication needed to support the computation. Several functions are necessary to implement this advanced function. The functions are noted one by one.

(i) Whole jobs are generated at participating nodes, and must be subdivided into workable pieces (termed here partial jobs) before they can be distributed.

(ii) A load balancing function is crucial: one wants to effectively utilize available capacity by holding the local load as close to capacity as possible.

(iii) Results for partial jobs (belonging to a single whole job), obtained at a large number of nodes on the network, need to be gathered in some fashion that eventually transports them back to the initiating node.

Function (i) here is carried out at a single node. Function (ii), in contrast, is a good candidate for application of CAS-like mechanisms. Function (iii) is likely best coordinated with (ii): the gathering function basically must invert the distribution that was done in (ii).

Distributed storage

Here the capacity to be shared is storage capacity. This is the principal difference from the previous function. Distributed storage may also be thought of as a utility, in this case analogous to distributed warehousing: participating nodes seek to utilize efficiently their shared pool of data storage capacity. The more basic functions needed to implement a distributed storage system differ somewhat from the previous list. One major reason for this is that, with distributed storage, the gathering step is initiated by the user, rather than by the nodes providing the capacity. And the user will likely not want to wait to retrieve the warehoused data. This means that the *reliability of on-demand retrieval* will be an important consideration here—in contrast to the distributed processing case (where retrieval is typically not on-demand by the source).

Some basic functions involved are:

(i) Subdivision (of files, data sets, etc) is still needed, although here the task is relatively simple.

(ii) Load balancing is again the crucial, and challenging, task.

(iii) Because of the need for reliable on-demand retrieval—along with the unreliability of the individual nodes—there needs to be some strategy and mechanisms for replication of the stored results. This strategy may seek to employ information, in the form of (for example) statistical correlations of downtimes among the various nodes. Gathering such information is a distinct function (in fact, an example of collective computation).

(iv) Retrieval of the distributed data is also of course a needed function.

It seems clear that functions (ii)–(iv) need to be coordinated, while (i) is again a single-node function.

Load balancing

As seen from the description of the above two services, load balancing forms a very important backbone for the implementation of the 'services'. The criterion for an effective load balancing function is not only ensuring even utilization of the capacity of the system, but also accomplishing it fast enough. Moreover, a load balancing algorithm should also consider the unreliability of the peers participating in the network and model the algorithms according to it.

Distributed content

The distributed capacities utilized here are storage and bandwidth. The utility analogy is that of a distributed library, rather than a distributed warehouse. That is: the stored material is not "owned" by any one node—instead, the most essential sort of sharing involved is that of content or information. Content may be injected into the library by any participating node, and requested by any other. Hence the seeking node normally has no historical connection to the sought resources. Therefore, the most important/challenging task is the "retrieval" task.

Furthermore, the specification of the sought resources is typically much broader in the case of the library than is the case for the warehouse. In the latter case, the seeking node wants a specific object (whose placement was initiated by that node). In the former (library) case, one

may seek a specific object (which other nodes have placed on the net). Or, one may seek many objects (hits), specified in a broad way by criteria such as keywords.

For these reasons, search over a distributed information base, built up via decentralized contributions, is a challenging task. Furthermore, it is of great practical interest. Because of this combination of interest and challenge, a great deal of work has been addressed to this problem. Also, a number of supporting functions have been invoked.

Some of the basic functions which may be used to support this advanced function are:

(i) Indexing. It is likely to be useful to store, at some or all nodes, not only content, but also information about where other content may be found. Indexing is frequently used in current peer-to-peer content-sharing systems—both working systems (e.g. Gnutella [30]), and those under study in research.

(ii) Again, a replication strategy may be employed. A good strategy will place copies of content in a way that facilitates subsequent search. A working example of an effective replication strategy is that used in the Freenet [17] system.

(iii) One needs to route queries (Q routing) from an initiating node, in a way that efficiently finds satisfactory content without overloading the network's links.

(iv) Each node must test for search criterion satisfaction, so that it can decide how many hits (if any) are found in its content (or index).

(v) Information about hits must be routed back to the source of the query ('Reply routing' or R routing).

(vi) Testing for the termination of the search (via sufficient number of hits, or expiration) must be done. This function is easily implemented if the queries do not branch. If they do, some coordination is required. This function, plus (iii) through (v), make up the "basic" search function.

(vii) Retrieval of a selected subset of the hits is also needed.

Moreover, there can be some additional desirable function.

(viii) A recommendation function may be used to allow information, which does not strictly satisfy the presented search criteria, to be presented to the searching node. The recommendation may be thought of as arising from the network itself—i.e. from meta-information, in the form of structure or correlations present in the stored content.

2 State of the Art

In this section, we present a brief overview of the state of the art about the three advanced services - distributed storage, distributed processing and distributed contents. The review is however not exhaustive keeping in mind the huge literature available and the plethora of approaches adopted to develop scalable algorithms. Moreover, many of the works are targeted to

achieve multiple goals. Thus, it is difficult to differentiate them for a particular service. Therefore, in the following survey, when we are reporting about some algorithm/schema/protocol pertaining to some services, we are simply implying it to be most effective for that service.

A vast repository of literature illustrating both the research and commercial progress of grid computing is available. We are not summarizing that portion of development. Although distributed processing is synonymous to grid computing, the spirit of BISON is to deal with dynamical systems, with peers which are unreliable, with topology which is constantly changing and peers which are not owned by any single organization/consortium. This is not a classic case of grid computing. In a grid computing environment, the peers generally agree to collaborate for a particular job and guarantee some reliability. In a dynamic environment, peers although may have agreed to collaborate for some jobs but the agreement is very 'loose' and they are only ready to perform when they don't have any 'real' work. So although we have a lot to learn from grid computing, essentially our perception of distributed processing is slightly different. A very good survey on grid computing is available in [2]. In this section, we separately report the state of art of distributed processing, distributed storage and distributed content. While reporting the state of the art about distributed processing, we only present a survey of load balancing schemes which form the backbone of distributed processing. The other related issues like reliability, routing etc. are more or less covered when we present the state of the art of distributed storage.

Distributed processing and load balancing: The efficiency of load balancing is interpreted both (i) from the system's point of view and will be ensured through the fair load distribution across all peers in the system, and (ii) from a user's point of view, by facilitating short response times to user requests. The basic challenge which lies ahead is achieving these goals through distributed algorithms - algorithms which don't have any global information at their disposal. A brief review of some of the schemes is given.

Looking at the literature, one discovers that load balancing is a problem that has been studied in detail in the field of distributed computing. Unfortunately, many of the results are not directly applicable to the field of p2p systems. The reason is that many of these results ([19, 20, 39], but the list could be very long) are often limited to static networks (meaning either with the load defined at the beginning of the computation, or systems with no failures). There are however a few notable exceptions like [1], where load-balancing is obtained by randomly selecting two nodes and moving load from one to another, if necessary. The algorithm can be executed in both synchronous and asynchronous mode. For unstructured p2p networks, the authors in [53] use fairness criteria to distribute load across nodes. Fairness is determined both in terms of the amount of processing a particular job needs and the computational capacity of the node involved. The authors show that their algorithm takes care of differing processing power and transient nodes. Moreover, the algorithm can be used to balance load for distributed storage and content.

The system Anthill, builds p2p systems using ideas from nature-inspired computing and develops load balancing applications [40]. In this paper, ants drop pheromone to set the path from highly loaded nodes to lightly loaded nodes which ensures fast transfer of load.

More interesting is the fact that the load-balancing problems are an open issue in *distributed hash table* (DHT), where the load may not always be evenly spread over the nodes. This can lead to hotspots and degraded performance. In particular, flash crowds, when sudden large spurts

of offered load are targeted for a particular item or service of interest, can easily overwhelm individual nodes. For these reasons, DHT should be able to adaptively rebalance the load when a temporary imbalance arises. Such adaptive algorithms must respond quickly to flash crowds and other temporary overloads, but must also not incur much overhead during more routine operations, and must be able to cope with node heterogeneity. Two papers [4, 43] provide two initial approaches for load balancing for DHTs. The scheme proposed in [43], uses the concept of virtual server transfer to balance load. Virtual Server represents the contiguous region of the identifier space of a hash table and is considered as unit of load. The scheme uses the concept of light and heavy nodes and develop greedy algorithms to transfer load from light to heavy nodes. Three mappings between light and heavy nodes are considered. The mappings are respectively one to one, one to many and many to many, whereby one or multiple nodes are used to transfer load from heavy nodes. The authors show that the algorithms fairly successfully (80 - 95%) achieve the goal of ultimately making all the nodes light. The work in [49] addresses the problem of load balancing in order to face flash crowds in a static system architecture and assumes global knowledge.

Distributed storage: Today's exponential growth in network bandwidth, storage capacity, and computational resources has inspired a whole new class of distributed, peer-to-peer storage infrastructures. Systems such as Farsite [3], Freenet[17], Intermemory [5], OceanStore [32], CFS [10], and PAST [15] seek to capitalize on the rapid growth of resources to provide inexpensive, highly-available storage without centralized servers.

The standard requirements which distributed storage is dealing with are as follows: *Availability:* Information can be accessed 24 hours a day, seven days a week; *Durability:* Information entered into the system will last virtually forever; *Access control:* Information is protected. Access control includes privacy (unauthorized entities cannot read information) and write-integrity (unauthorized entities cannot change information); *Authenticity:* Adversaries cannot substitute a forged document for a requested document; *Denial-of-service (DOS) resilience:* It is difficult for an adversary to compromise availability.

Moreover, several interesting desirable goals are also discussed recently: *Massive scalability:* The system should work well with thousands, millions, or even billions of clients; *Anonymity:* It is impossible or very difficult for an outside observer to ascertain who has produced a document and who has examined it; *Deniability:* Users should be able to deny knowledge of data stored on their machines. *Resistance to censorship:* No one can censor information once it is written to the system [42]. It is an open question whether all of these properties can coexist.

However, till now, availability and durability remain the fundamental focus of research. The designers of these systems propose to achieve high availability and long-term durability, in the face of individual component failures, through replication and coding techniques. Wide-scale replication, although increases availability and durability, introduces two important challenges to system architects. First, system architects must increase the number of replicas to achieve high durability for large systems. Second, the increase in the number of replicas increases the bandwidth and storage requirements of the system. Excessive replication can incur high storage and bandwidth overhead. Thus, several p2p systems, such as Intermemory, OceanStore and FreeHaven, have utilized an efficient form of redundancy called erasure coding in which each chunk of data is transformed into many fragments. The essential property of this transformation is that only a fraction of the fragments must be recovered to reconstruct the data

[22]. The availability also introduces the problem of retrieving the data on demand to the peer which has stored them. This is conceived as a routing problem. Several protocols like Pastry [15] and Tapestry [23] are proposed to tackle the problems. The protocols are also taking care of the fact that there will be node failures which are potentially capable of damaging the routes.

Many p2p storage systems are repositories of read-only information. The biggest barrier to providing a writable system is establishing consistency about the identity of the latest copy of data. To address such active decision making as consistency establishment among nodes, several recent systems, such as OceanStore and Farsite [3], have employed Byzantine Agreement [33]. Byzantine Agreement allows a set of peers to come to a unified decision about something, even if some of them (less than one third) are actively attempting to compromise the process. Should the correct number of nodes agree, the result can even be signed in aggregate with threshold signatures [32] to permit others to verify the decision at a later date.

However, an important limitation of all the models is that they are built upon the belief that components fail independently. When this assumption is violated, many purported guarantees are lost. For instance, replica placement schemes do not protect data when servers holding replicas fail together (are correlated). As another example, Byzantine Agreement algorithms do not function when many servers are corrupted simultaneously. Unfortunately, correlations exist in all real systems. Peers may share the same subnet, owner, software release, operating system, or geographic location. Most p2p systems rely on random component placement and increased redundancy to combat correlated failures. While effective, these heuristics are not the best way to avoid correlations. A more rigorous search is required and we can learn from nature and society how to readjust in case of a series of correlated failures.

Distributed content: The distributed content and its associated search algorithms should maximize the goals of efficiency in terms of resources (bandwidth, storage facility, processing power), quality of service (number of results, response time) and robustness. Side by side it should provide the peers more autonomy (in terms of with whom it wants to be neighbor and/or in which machines it is ready to share its content) and expressiveness (how differently can a user express his search). Different protocols which have been developed are trying to maximize these interests. A brief sketch is provided.

One of the most recent models of peer-to-peer systems, as described in the survey about p2p systems from HP [38], is the document routing paradigm. This paradigm is now the predominant model for many kinds of p2p applications [6, 10, 15, 26, 32, 46, 50]. All these novel applications are based on the concept of *distributed hash tables*, or DHTs. Examples of DHTs include Chord [51], CAN [44], Pastry [45], Tapestry [23], Viceroy [36], and Koorde [29]. Alternative names are distributed lookup services, location-independent routing services, structured overlay networks, etc.

In the DHT model, each object of interest (either documents in a file-sharing application [15], blocks of data in a storage-sharing application [10], "topics" in a publish-subscribe application [46], etc.) is assigned a key in a m-bit virtual address space. This key may be obtained in several ways, such as hashing the name of a document in a file-sharing application, or mapping the blocks of a distributed file system in a storage-sharing application. The virtual address space is partitioned into cells, which form contiguous regions of this address space. Depending on the design of the specific DHT algorithm, either a single host or a set of hosts is assigned to each cell of the virtual address space. Each host is assigned to one or more cells, and main-

tains copies of those key-value bindings whose key values lie within its assigned cells. The partitioning of the address space and the assignment of hosts to cells is dynamic and, in particular, changes whenever a node enters, departs or fails, or whenever load balancing is required to match the capabilities of nodes with the bandwidth, storage and processing requirements assigned to them.

These systems maintain highly structured overlay infrastructures that rely on a symmetric distributed design. However, they can reach Internet scales without incurring undue overhead. There is one basic operation, lookup(key), which returns the identity (e.g. the IP address) of the node storing the object with that key. This operation allows nodes to put and get files based on their key, thereby supporting a hash-table-like interface. The lookup() operation typically requires only O(log n) steps, where n is the number of nodes to find a search item. However, although it is quite fast, its ability to operate with extremely unreliable nodes has not yet been demonstrated. Moreover, the systems built upon distributed hash tables cannot deal with partial-match queries (e.g., searching for all objects whose titles contain two specific words).

In comparison to highly structured overlay network maintained by DHT, the loosely structured p2p networks, Freenet [17], Free Haven [14], Mojo Nation [37], are extremely robust but the search time is quite slow. Flooding and random walk are two of the methods used to perform search in these networks. In order to improve the performance, the loosely structured networks employ different schemes to replicate data, so as to develop a correlation between the network structure and data and consequently enhance the speed of search. There is Freenet which replicates data as soon as they are searched. However, the replication mechanism needs not be initiated by search. For example FastTrack [16] takes more proactive measures of replicating the data even if not searched. FastTrack utilizes the heterogeneity of the nodes capacity and bandwidth to store indices more and more on the high bandwidth nodes (also termed as superpeers). It is believed that Kazaa [31] is also following the same mechanism (Since, all the algorithms of Kazaa are proprietary in nature, the exact features of the algorithms are not known). Different systems use different replication strategies like proportional replication, uniform replication and square root replications [35]. However, still now, none of the replication strategies has been proven to be the best. There is a wide scope of research in this area. Different biological processes can be explored in this area.

Moreover, there are works on semantic overlay networks (SON) which propose clustering nodes on basis of the same semantics [47]. That is, nodes having identical semantics form neighbors to each other. The semantics can consider the network to be a flat structure [8, 52] as well as a super-peer structure [34]. There are many challenges when building SONs. First, we need to be able to classify queries and nodes. We need to decide the level of granularity for the classification (e.g., just rock songs versus soft, pop, and metal rock) as too little granularity will not generate enough locality, while too much would increase maintenance costs. We need to decide when a node should join a SON (if a node has just a couple of documents on rock, do we need to place it in the same SON as a node that has hundreds of rock documents?). Finally, we need to choose which SONs to use when answering a query.

3 BISON Models

During the previous year of the BISON project, there have been several initial attempts to develop advanced services for p2p networks. The works include exploring different load balancing algorithms and developing an effective strategy to search distributed contents in $p2p$ networks. Moreover, there has been a proposal to build up system model which will integrate the basic and advanced functions and also simultaneously re-enforce each other's efficiency. The load balancing protocol is being explored in two different ways by Bologna and Telenor. The algorithms related to distributed contents are primarily developed by Dresden. The system approach has been propounded by researchers in Bolgna. We are noting them one by one.

3.1 Modular Approach for Building Advanced Functions

One important dimension along which the workplan of the BISON project is layed out is the dichotomy of basic and advanced services. Advanced services are envisioned not only to be simply more complicated than basic services but they also rely and build upon basic services. In this section we elaborate on this difference emphasizing the possibility of a modular approach to design fully distributed self-organizing applications.

The promises of p2p technology have already been partially fulfilled by numerous applications. Unfortunately, the underlying protocols on which they are built are often complex and unpredictable. Their behavior is not fully understood, and often, can be explained only in terms of the theory of complex networks or dynamic systems. Given the lack of traditional hard guarantees regarding expected outputs, users outside the scientific community—especially engineers and application developers—might experience difficulties in exploiting the growing body of available knowledge.

In our opinion, a more significant exploitation of p2p technology requires a *modular paradigm* where well-understood and predictable components with clear interfaces can be combined to implement arbitrarily complex functions. The goal of this section is to report on our ideas and initial results towards this objective.

The first step in our advocated paradigm is to identify a collection of primitive components, that is, simple p2p protocols that can be used as *building blocks* for constructing more complex protocols and applications. An informal classification of these building blocks in two broad classes is possible:

Overlay protocols maintain connected communication topologies over a set of nodes. We refer to such topologies as *overlays*, as they are built over underlying networks like the Internet. An example is NEWSCAST [27], that maintains a random overlay.

Functional protocols are aimed at implementing basic functions for other components. An example for such a function is *aggregation* [54], a collective name for functions that provide global information about a distributed system. These functions include finding extremal values, computing averages and sums, counting, etc.

A modular approach offers several attractive possibilities. It allows developers to plug different components implementing a desired function into existing or new applications, being certain

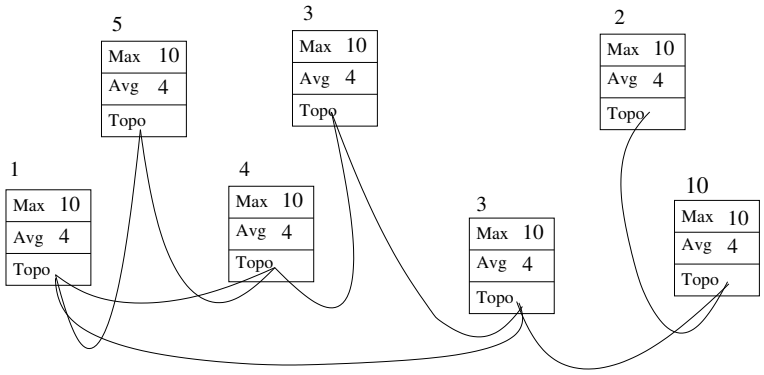


Figure 1: A simple example network. The environment consists of a set of numeric values, one at each node. Two protocols, MAX and AVG, find the maximum and the average of these values, respectively. They are based on protocol TOPO, whose task is to maintain an overlay topology (represented by the connections between nodes).

that the function will be performed in a predictable and dependable manner. An even more interesting possibility is to combine building blocks to form more complex applications that perform relatively sophisticated functions like file sharing or load balancing.

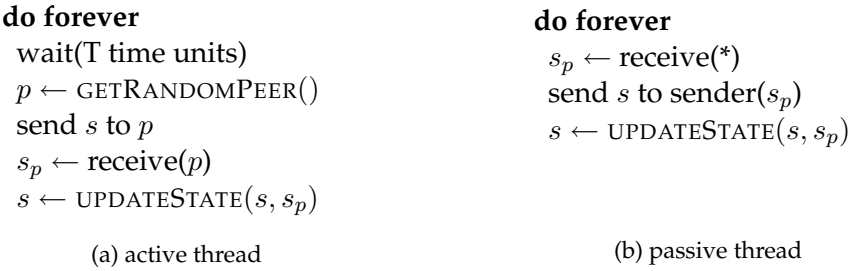
Building blocks must be simple and predictable, as well as extremely scalable and robust. In this way, research can focus on self-organization and other important emergent features, without being burdened by the complexity of the protocols. Our building blocks that are described in deliverable D05 are typically no more complicated than a cellular automaton or a swarm model which makes them ideal objects for research. As a result, practical applications can also benefit from a potentially more stable foundation and predictability, a key concern in fully distributed systems.

3.1.1 System Model

Figure 1 illustrates our system model. We consider a network comprising a large collection of *nodes* that communicate through the exchange of messages and are assigned unique identifiers. The network is highly dynamic (new nodes may join and old ones can leave at any time) and subject to failures (nodes may fail and messages can be lost).

Each node runs a set of *protocols*. Protocols can be standalone applications, or may provide some service to other protocols. Each protocol instance may communicate with other protocols located at the same node (e.g., to import or export a service) and with other instances of the same protocol type located at remote nodes (e.g., to implement a function).

We assume that nodes are connected by an existing *physical network*. Even though the protocols we suggest can be deployed on arbitrary physical networks, including sensor and ad-hoc networks, in the present work we consider only fully connected networks, such as the Internet, where each node can (potentially) communicate with every other node. In this way, arbitrary overlay topologies may be constructed, and a functional protocol may deploy the most appropriate overlay for implementing its functions.



(a) active thread

(b) passive thread

Figure 2: The skeleton of a push-pull epidemic-style protocol. Notation: s is the state of this node, s_p is the state of the peer p.

The physical network provides only the possibility of communication. To actually communicate with its peers, a node must know their identifiers. At each node, the task of an overlay protocol is to collect and maintain up-to-date identifiers in order to form a connected topology with some desired characteristics. Given the large scale and the dynamicity of our envisioned system, these collections are normally limited to a rather small subset of the entire network.

Apart from communicating with other peers, protocols may also interact with their *environment*. Any input that originates from outside the protocol set falls into this category. The environment may include user interactions, sensor information, and any application-specific data such as load in a load balancing system and free space in a distributed storage system.

In our model, modularity is implemented at the level of protocols. Protocols must provide and implement well-defined interfaces, in order to allow developers to plug different implementations of the same protocol into their applications. For example, as explained in the next section, the aggregation protocols illustrated in Figure 1 make use of an overlay protocol to communicate with peer nodes so as to compute the aggregates. Any implementation of overlay can be used, as long as it provides standard interfaces and a topology with the appropriate characteristics.

3.1.2 Example Building Blocks

Let us briefly mention two building blocks described in deliverable D05, NEWSCAST and *epidemic-style aggregation*. Additional information can be found in the related papers [27, 28]. In Section 3.2 we show the importance of these basic building blocks, when *aggregation* is used to evaluate a more complex function: *load balancing*.

Both protocols are based on the push-pull epidemic-style scheme illustrated in Figure 2. Each node executes two different threads. The *active* one periodically initiates an *information exchange* with a peer node selected randomly, by sending a message containing the local state and waiting for a response from the selected node. The *passive* one waits for messages sent by an initiator and replies with its local state. The term push-pull refers to the fact that each information exchange is performed in a symmetric manner: both peers send and receive their states.

Method UPDATESTATE builds a new local state based on the previous local state and the state

received during the information exchange. The output of UPDATESTATE depends on the specific function implemented by the protocol. The local states at the two peers after an information exchange are not necessarily the same, since UPDATESTATE may be non-deterministic or may produce different outputs depending on which node is the initiator.

The period of wall clock time between two consecutive information exchanges is called the *cycle length*, and is denoted by T. Even though the system is not synchronous, it is often convenient to talk about *cycles* of the protocol, which are simply consecutive wall clock time intervals of length T counted from some convenient starting point.

In the case of NEWSCAST the state is a fixed sized set of peer addresses and UPDATESTATE is a merging operation: the output of UPDATESTATE is the freshest addresses from the union of the two address sets with respect to creation date. This protocol results in an adaptive and robust unstructured overlay topology.

In the case of aggregation the state is a numeric value. For example, in the case of averaging UPDATESTATE returns the average of the two states. The protocol can be proven to exponentially converge to the true average of the values held by the nodes [28]. We use this protocol to develop load balancing algorithm in the next section.

3.2 Load Balancing Using a Modular Paradigm

Let us define the load balancing problem, which will be our example application for illustrating the modular design paradigm. We assume that each node has a certain amount of load and that the nodes are allowed to transfer all or some portions of their load between themselves. The goal is to reach a state where each node has the same amount of load. To this end, nodes can make decisions for sending or receiving load based only on locally available information.

Without further restrictions, this problem is in fact identical to the averaging problem described in Section 3.1.2. In a more realistic setting however, each node will have a limit, or *quota*, on the amount of load it can transfer in a given cycle (where *cycle* is as defined in Section 3.1.2). In our present discussion we will denote this quota by Q and assume that it is the same for each node.

3.2.1 The Optimal Algorithm

For the sake of comparison, to serve as a baseline, we give theoretical bounds on the performance of any load balancing protocol that has access to global information.

Let $a_{i,1}, \ldots a_{i,N}$ represent the individual loads at cycle i, where N is the total number of nodes. Let μ be the average of these individual loads over all nodes. Note that the global average does not change as a result of load transfers as long as work is "conserved" (there are no node failures). Clearly, at cycle i, the minimum number of additional cycles that are necessary to reach a perfectly balanced state is given by

$$\max_j \left\lceil \frac{|a_{i,j} - \mu|}{Q} \right\rceil \tag{1}$$

Let $a_{i_1}, \ldots, a_{i_N}$ be the decreasing order of load values $a_1, \ldots, a_N$
$j \leftarrow 1$
while ($a_{i_j} > \mu$ and $a_{i_{N+1-j}} < \mu$)
$a_{i_j} \leftarrow a_{i_j} - Q$
$a_{i_{N+1-j}} \leftarrow a_{i_{N+1-j}} + Q$
$j \leftarrow j + 1$

Figure 3: One cycle of the optimal load balancing algorithm. Notation: μ is the average load in the system, N is the network size, Q is the quota.

and the minimum amount of total load that needs to be transferred is given by

$$\frac{\sum_j |a_{i,j} - \mu|}{2}. \tag{2}$$

Furthermore, if in cycle i all $a_{i,j} - \mu$ ($j = 1, \ldots, N$) are divisible by Q, then the optimal number of cycles and the optimal total transfer can both be achieved by the protocol given in Figure 3. This algorithm is expressed not as a local protocol that can be run at each node, but as a global algorithm operating directly on the list of individual loads. It relies on global information in two ways. First, it makes a decision based on the overall average load (μ) which is a global property and it relies on globally ordered local load information to select nodes with specific characteristics (such as over- or under-loaded) and for making sure the quota is never exceeded.

It is easy to see that the total load transferred is optimal, since the load at each node either increases monotonically or decreases monotonically, and when the exact global average is reached, all communication stops. In other words, it is impossible to reach the balanced state with any less load transferred.

The algorithm also achieves the lower bound given in (1) for the number of cycles necessary for perfect balance. First, observe that during all transfers exactly Q amount of load is moved. This means that the property that all $a_{i,j} - \mu$ ($j = 1, \ldots, N$) are divisible by Q holds for all cycles, throughout the execution of the algorithm. Now, we only have to show that if $\max_j |a_{i,j} - \mu| = k \cdot Q \geq 0$ (k is an integer) then

$$\max_j |a_{i,j} - \mu| - \max_j |a_{i+1,j} - \mu| = Q. \tag{3}$$

To see this, define $J = \{j^* | \max_j |a_{i,j} - \mu| = |a_{i,j^*} - \mu|\}$ as the set of indices which belong to nodes that are maximally distant from the average. We have to show that for all nodes in J, a different node can be assigned that is on the other side of the average. We can assume without loss of generality that the load at all nodes in J is larger than the average because (i) if it is smaller, the reasoning is identical and (ii) if over- and under-loaded nodes are mixed, we can pair them with each other until only over- or under-loaded nodes remain in J. But then it is impossible that the nodes in J cannot be assigned different pairs because (using the definition of J and the assumption that all nodes in J are overloaded) the number of under-loaded nodes has to be at least as large as the size of J. But then all the maximally distant nodes got their load difference reduced by exactly Q, which proves Eq (3).

Motivated by this result, in the following we assume that (a) the initial load at each node is an integer value, (b) the average is also an integer and (c) we are allowed to transfer at most one

```
do forever
  q ← Q
  wait(T time units)
  μ ← GETAVERAGELOAD()
  if (q = 0) continue
  if (|a − μ| < Q) FREEZE()
  if (a < μ)
    p ← GETOVERLOADEDPEER(q, μ)
    if (p ≠ null) TRANSFERFROM(p, q)
  else
    p ← GETUNDERLOADEDPEER(q, μ)
    if (p ≠ null) TRANSFERTO(p, q)
```

(a) active thread

```
GETOVERLOADEDPEER(q, μ)
  (p_1, ..., p_c) ← GETNEIGHBORS()
  Let p_{i_1}.a, ..., p_{i_c}.a be the decreasing
    order of neighbor load values p_1.a, ..., p_c.a
  for j = 1 to c
    if (p_{i_j}.a > μ and p_{i_j}.q ≥ q)
      return p_{i_j}
  return null

GETUNDERLOADEDPEER(q, μ)
// Defined analogously
```

(b) peer selection

Figure 4: A modular load balancing protocol. Notations: a is the current load, Q is the total quota, q is the residual quota and c is the number of peers in the partial view as determined by the overlay protocol.

unit of load at a time. This setting satisfies the assumptions of the above results and serves as a tool for simplifying and focusing our discussion.

3.2.2 A Modular Load Balancing Protocol

Based on the observations about the optimal load balancing algorithm, we propose a protocol that is based purely on local knowledge, but that approximates the optimal protocol extremely well, as we show in Section 3.2.4.

Figure 4 illustrates the protocol we propose. The basic idea is that each node periodically attempts to find a peer which is on the "other side" of the global average and has sufficient residual quota. If such a peer can be found, load transfer is performed.

The approximation of the global average is obtained using method GETAVERAGELOAD, and the peer information is obtained using method GETNEIGHBORS. These methods can be implemented by any appropriate component for average calculation and for topology management.

We assume that in each cycle, each node has access to the current load and residual quota of its peers. This latter value is represented by local variable q at each node, which is initialized to Q at the beginning of each cycle and is updated by decrementing it by the actual transferred load. This information can be obtained by simply asking for it directly from the peers. This does not introduce significant overhead as we assume that the load transfer itself is many orders of magnitude more expensive. Furthermore, as we mentioned earlier, the number of peers is typically small ($c = 20$ is typical).

Note that once the local load at a node is equal to the global average, the node can be excluded from future considerations for load balancing since it will never be selected for transfers. By excluding these "balanced" nodes, we can devote more attention to those nodes that can benefit

```
do forever
  q ← Q
  wait(T time units)
  if (q = 0) continue
  p ← GETPEER(q, a)
  if (p.a < a) TRANSFERTO(p, q)
  else TRANSFERFROM(p, q)
```

(a) active thread

```
GETPEER(q, a)
  (p_1, ..., p_c) ← getNeighbors()
  Let p_{i_1}.a, ..., p_{i_c}.a be the decreasing
    order of neighbor load values p_1.a, ..., p_c.a
    according to the ordering defined by
    |a − p_1.a|, ..., |a − p_c.a|
  for j = 1 to c
    if (p_{i_j}.q ≥ q) return p_{i_j}
  return null
```

(b) peer selection

Figure 5: The basic load balancing protocol. Notations: a is the current load, Q is the total quota, q is the residual quota and c is the number of peers in the partial view as determined by the overlay protocol.

from further transfers. The protocol of Figure 4 implements this optimization through the method FREEZE. When a node executes this method, it starts to play "dead" towards the overlay protocol. As a result, the node will be removed from the communication topology and the remaining nodes (those that have not yet reached the average load) will meet each other with higher probability. In other words, peer selection can be more efficient in the final phases of the execution of the balancing protocol when most nodes already have reached the average load. Although the optimization will result in a communication topology that is partitioned, the problem can easily be solved by adding another overlay component that does not take part in load balancing and is responsible only for maintaining a connected network. Note also that the averaging component uses the same overlay component that is used by the load balancing protocol.

A key feature of the averaging and overlay protocols is that they are potentially significantly faster than any load balancing protocol. If the quota is significantly smaller than the variance of the initial load distribution, then reaching the final balanced state can take arbitrarily long (see Equation (1)). On the other hand, averaging converges exponentially fast as described in Deliverable D05. This fact makes it possible for load balancing to use the approximation of the global average as if it were supplied by an oracle with access to global information. This scenario where two (or more) protocols operate at significantly different time scales to solve a given problem is encountered also in nature and may characterize an interesting general technique that is applicable to a larger class of problems.

3.2.3 A Basic Load Balancing Protocol

In order to illustrate the effectiveness of using the averaging component, we suggest a protocol which does not rely on the average approximation. The protocol is shown in Figure 5.

This protocol attempts to replace the average approximation by heuristics. In particular, instead of choosing a peer from the other side of the average, each node picks the peer which has a maximally different load (larger or smaller) from the local load. The step which cannot

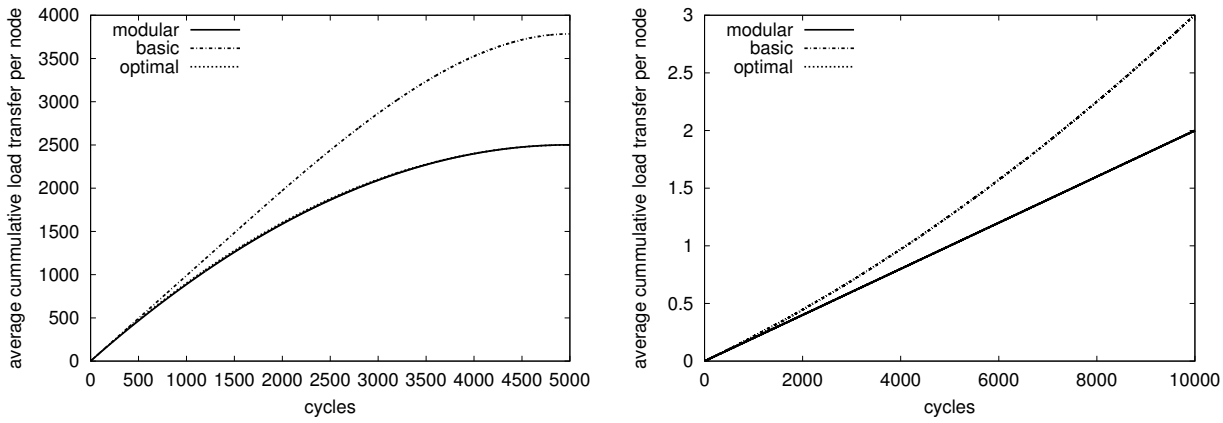
be replaced however is the FREEZE operation. Performing that operation depends crucially on knowing the global average load in the system.

3.2.4 Empirical Results

Empirical studies have been performed using PEERSIM, the simulator we developed and described in deliverable D11. We implemented the three protocols described above: the optimal algorithm, the modular protocol that is based on the averaging protocol and NEWSCAST and the basic protocol that has no access to global average load. As components, the methods of Figure 4 were instantiated with the aggregation protocol of Section 3.1.2 for averaging and NEWSCAST for the overlay.

In all our experiments, the network size was fixed at $N = 10^4$ and the partial view size was $c = 40$. We examined two different initial load distributions: linear and peak. In the case of linear distribution, the initial load of node i ($i = 0, \ldots, N$) was set to exactly $i - 1$ units. In the case of peak distribution, the load of exactly one node was set to 10^4 units while the rest of the nodes had no initial load. The total quota for load transfer in each cycle was set to one load unit ($Q = 1$).

During the experiments the variance of the local load over the entire network was recorded along with the amount of load that was transferred during each cycle. We do not show the data on variance—which would give information about the speed of reaching the balanced state—because all three protocols have identical (i.e., optimal) convergence performance for both initial distributions.



(a) linear load distribution

(b) peak load distribution

Figure 6: Cumulative average load transferred by a node until a given cycle in a network of size 10^4. The curves corresponding to the optimal algorithm and the modular protocol overlap completely and appear as a single (lower) curve. The final point in both graphs (5000 and 10000 cycles, respectively) corresponds to a state of perfect balance reached by all three protocols.

Figure 6 presents results for total load transferred during the execution of the three solutions.

Each curve corresponds to a single execution of a protocol, as the variance of the results over independent runs is diminishing. As can be seen from the figures, the load transferred by the modular protocol is indistinguishable from the amount that is optimal for perfect balancing in the system.

3.3 Load Balancing Using Diffusion, Chemotaxis and Ants

Here the aim is to test two well known biological concepts against simple classical diffusion, for efficient load balancing. Diffusion is included as a reference case: it is a decentralized algorithm, but without any claim to swarm intelligence or emergence of any kind. Diffusion is also a well-studied method for load balancing in parallel processing applications [7, 9, 12, 13, 24, 25, 48, 55, 56]. Typical works in this direction focus on regular topologies; in contrast, BISON seeks methods which work well in dynamic topologies, which are not necessarily regular. In this section, we assume no knowledge of the topology, and that no global information (such as the average load) is available.

We want to carry out a "fair" comparison of three distinct mechanisms, in an attempt to answer the question: Which is the most "smart" (efficient) system of all? The concept of "fairness" will be specified in detail below; the principal idea is that each system should have the same basic "brute force" ability to move load, so that a more efficient (ie, faster) approach to load balancing is achieved through "smartness" rather than brute force. The three schemes for load balancing—through diffusion, using ant algorithms, and using chemotaxis—are elaborated below. Since the schemes are decentralized, algorithms are given for arbitrary node i. All the nodes execute the same algorithm.

3.3.1 Diffusion

For all models in this section, including diffusion, the load at node i is denoted ϕ_i. Also, each node is assigned a *capacity* C_i. The aim of load balancing then is to seek an even distribution of the difference quantity $\phi_i - C_i$. Ideally, this difference would be zero everywhere. However in some cases, and in the models considered here, the total load is a constant—there is no mechanism modeled for creating or eliminating load—and the node capacities, and hence the total capacity, is also constant. For reasons given below, these two constants will be set equal.

The discrete version of the diffusion equation is

$$\Delta\phi = c \cdot L \cdot \phi \tag{4}$$

where $\Delta\phi$ is the discrete-time change in the vector of loads ϕ, c is a diffusion constant, and L is the Laplacian matrix; or equivalently

$$\Delta\phi_i = c \cdot \Big(\sum_{j=nn(i)} \phi_j - k_i \cdot \phi_i \cdot \Big). \tag{5}$$

Here k_i is the degree of node i, and $nn(i)$ denotes the nearest neighbors of i. The average over i's neighbors may be written as

$$\phi_{av,i} = \frac{\sum_{j=nn(i)} \phi_j}{k_i}; \tag{6}$$

hence we see that our diffusion rule gives no change in load at i when $\phi_i = \phi_{av,i}$.

One way of decomposing the discrete diffusion equation above, is as follows. At a given time, we define the change in load moving from i to j as $\Delta\phi_{i\to j}$. Then we can obtain the above diffusion rule by setting

$$\Delta\phi_{i\to j} = c \cdot \phi_i \tag{7}$$

where c is the same diffusion constant. Now we want to get the total movement of load *out* from node i at a given time. This is

$$\Delta\phi_{i,out} = \sum_{j=nn(i)} \Delta\phi_{i\to j} = c \cdot k_i \cdot \phi_i. \tag{8}$$

This equation may be viewed as the basic equation for discrete diffusion (although, in fact, only the *net* change in ϕ_i has any independent meaning in these models). We focus on this equation because we will use it to quantify our "fairness criteria" for comparing the various models here. That is, a fair comparison should give each system the same "strength" or "brute force ability" for moving load; then the more efficient system or algorithm will achieve its advantage from "smart" movement of load, rather than brute force. We believe that it is reasonable to take something like Eq. (8) as defining the brute-force load-moving ability for diffusion. Our goal will then be to define promising models for efficient load balancing, and then to ensure that each such model will have, as nearly as possible, a brute force load-moving ability equal to that found here for diffusion.

Finally, we must correct the above equations for capacity. Eq. (7) becomes

$$\Delta\phi_{i\to j} = c \cdot (\phi_i - C_i) \tag{9}$$

and Eq. (8) becomes

$$\Delta\phi_{i,out} = c \cdot k_i \cdot (\phi_i - C_i). \tag{10}$$

Similar corrections for the other equations are easily made. Note that, even though the capacities C_i amount to a simple shifting of the zero point for the loads, they affect our measure of brute-force load-moving ability: the amount of load moved in Eq. (8) can differ significantly from that moved according to Eq. (10). Hence—since we will use capacities in the other models considered—we take Eq. (10) as our reference level for brute-force load-moving ability.

3.3.2 Chemotaxis (CT)

The basic idea here is to use a signal in order to accomplish the load-distribution task faster than diffusion. We illustrate this idea with a simple figure which shows the initial distribution of load on a one-dimensional plane.

Note the large load at the center of the simple, one-dimensional distribution. The standard diffusion mechanism will send load equally in both directions (left and right) from this site—even though it is clear from our global view that more load should be sent to the right. The diffusing system will only discover this after left-diffusing load from the large central piece collides with right-diffusing load from the left.

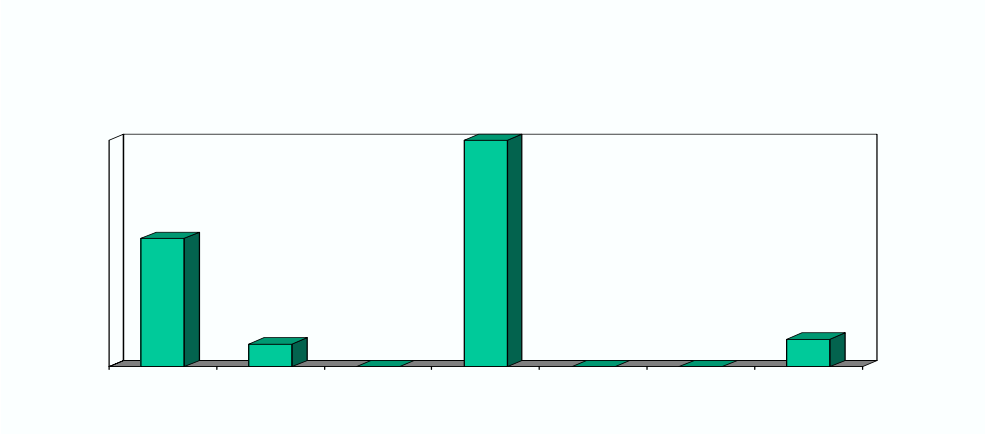


Figure 7: Distribution of Initial Load

This kind of inefficiency for standard diffusion is not just valid for networks which resemble a 1D (or higher-dimensional) space. That is, the *net* load transferred (flow) over each link, from a static starting distribution to convergence, is known to satisfy an optimality criterion; however in general the total movement of load is much larger than this net flow [12, 25]. One method for combating this inefficiency is to compute the flow using a diffusion algorithm, but without actually moving the load. Then, in the second phase of the procedure, one carries out the flow [11, 12]. This two-phase approach is suited for static problems. However, in our chemotaxis model, we will present an approach in the same spirit, but meant for dynamic load balancing problems. That is, we give the load a chemotactic *signal*, which is constantly emitted by the load. The signal diffuses much faster than the load could diffuse. The load itself moves down gradients of signal. Thus, if the signal diffuses quickly enough, it will guide the load in Figure 7, for example, more to the right than to the left—eliminating much of the wasted movement.

In a real network, movement of both signal and load is limited by the network's effective bandwidth. However, the signal represents a quantity of data on the order of bits; while the load may be considerably larger. Hence it is plausible that the signal could move more rapidly than the load. The existence of these two time scales—one for signal, and one for load—is a crucial aspect of this proposed load-balancing mechanism.

Now we express the above idea in equations. We again assign capacity C_i to each node i.

The signal emitted at node i, at each time instant, is

$$S_i^{emit} = c_2 \cdot (\phi_i - C_i). \tag{11}$$

Here we see the utility of assigning capacities to each node: the emitted signal can be either positive or negative. If there were no capacity terms, so that loads were always positive or zero, there would be only sources of signal, and no sinks. In a real chemotactic system, there are sinks: diffusion away to infinity, and consumption of signal by bacteria. In our closed network, with only sources, the average signal level must grow steadily with time. Even though the load is only sensitive to *gradients* of signal, it is impractical to have to manage ever-growing real numbers; and a rescaling of the zero level of signal is a global problem.

We will thus set the total load ϕ_{tot} equal to the total capacity C_{tot}. This ensures that the average signal level is zero at all times. This is a global constraint, and thus somewhat artificial. However it allows us to compare these load-balancing mechanisms on equal terms.

In a more realistic system, load can be created and destroyed. In such systems, the artificial constraint is removed. The use of signal, emitted from load, should however still be useful, in fact more so: the local signal level, along with its time derivative, can serve as a form of feedback to the nodes that may create or destroy load.

Equation (11), along with the constraint $\phi_{tot} = C_{tot}$, gives us a system with balance between sources and sinks.

The movement of the load from node i to j is guided by the difference in signal level between the two participating nodes i and j:

$$\Delta\phi_{i\to j} = c_3 \cdot (S_i - S_j). \tag{12}$$

This difference is simply the discrete form of the gradient.

Finally, the signal itself obeys standard diffusion:

$$\Delta S_{i\to j} = c_4 \cdot S_i. \tag{13}$$

Now we come to the question of two time scales. For a continuous system, to ensure that diffusion of signal is much faster than diffusion of load, we would set $c_4 >> c$. For discrete systems, the choice is much less simple. In fact, it is a current research problem to find the value for the diffusion constant which gives the fastest diffusion; see for example Refs. [9, 11, 55, 56]. In particular, the optimal diffusion constant (or constants—see below) depends on the topology, with analytical results only available for regular topologies. Also, in general knowledge of the eigenvalues of the Laplacian matrix is needed to find the optimal value [9, 11, 56], or even to find a good but suboptimal value [25].

These ideas are clearly not suitable for a nonregular and highly dynamic topology. Instead, we seek a way to ensure fast diffusion of signal, without global knowledge, and in the face of a dynamic topology. We recall our basic assumption: signal (unlike load) requires negligible bandwidth, and so we can send as much as we like in each time step. The problem is then, how much to send, so as to ensure that the resulting diffusion is indeed fast.

Cybenko [9] defines a diffusion matrix $\mathbf{M}$, obtained as follows. First one generalizes equation (5) (also replacing ϕ with S) to the form

$$\Delta S_i = \sum_j \alpha_{ij} \cdot (S_j - S_i); \tag{14}$$

that is, the fraction of the difference which is transferred between i and j can vary for different links ij. This more general ("anisotropic") diffusion law gives

$$S_i^{t+1} = \Big(1 - \sum_j \alpha_{ij}\Big) S_i^t + \sum_j \alpha_{ij} S_j^t, \tag{15}$$

which in turn may be written in matrix form as

$$\mathbf{S}^{t+1} = \mathbf{M}\mathbf{S}^t. \tag{16}$$

This anisotropic diffusion rule is weight-conserving if $\alpha_{ij} = \alpha_{ji}$. Also, one takes the off-diagonal elements $M_{ij} = \alpha_{ij}$ to be zero when there is no link between i and j, and $\alpha_{ij} \geq 0$ otherwise. Finally, the diagonal elements $M_{ii} = 1 - \sum_j \alpha_{ij}$ represent the fraction of signal S_i that is not transferred away in one time step. Cybenko, and others, wish only to work with nonnegative values; hence it is common to take $M_{ii} \geq 0$. All these constraints make $\mathbf{M}$ a doubly stochastic matrix—a type for which much is known.

A diffusion process driven by Eq. (16) converges to a uniform distribution under the conditions that (i) the induced network (formed by those links with $\alpha_{ij} \neq 0$) is connected, and (ii) *either* the network is nonbipartite, *or* at least one $M_{ii} > 0$. The rate of convergence is determined by the gap between the largest eigenvalue of $\mathbf{M}$ (which is 1) and the magnitude of the eigenvalue with second largest magnitude. The convergence condition (ii) ensures that $\lambda = -1$ (which would make the gap = 0) is not in the spectrum.

One implication of the constraints in the above two paragraphs is that one should never send out more signal than one finds at a node. This constraint, although apparently motivated by the wish to keep to positive signal (load) values, may be necessary for first-order diffusion systems as represented by Eq. (16); however it can be violated in other schemes [21].

One way to hold to this constraint is to set

$$\alpha_{ij} = \alpha_{ji} = \min(1/k_i, 1/k_j). \tag{17}$$

This choice fulfills all of Cybenko's criteria, without requiring any global knowledge of the network topology. In fact, it also adapts easily to changes in topology, since it only requires communication among neighboring nodes. We will call this scheme 'FCDL': 'fast Cybenko diffusion with a local criterion'. FCDL has not, to our knowledge, been tested; we present it as a simple, decentralized candidate for (possibly) fast signal diffusion.

One might be more greedy and send, at each time step, *all* signal that is found at a node i. That is, set

$$\Delta S_{i\to j} = S_i/k_i, \tag{18}$$

so that

$$\Delta S_{i,out} = S_i. \tag{19}$$

Let us call this scheme 'SOE' (send out everything). SOE does not conform to the general scheme of Cybenko, since it gives a rule

$$\Delta S_i = \sum_{j=nn(i)} \Big(\frac{S_j}{k_j} - \frac{S_i}{k_i}\Big); \tag{20}$$

this is not in the form of Eq. (14). Nevertheless this process is weight-conserving, and it converges. We can see this by normalizing the $\mathbf{S}$ vector so that the total signal $\sum_i S_i = 1$. Then we see that we can view S_i as a probability fluid, and that iterations of Eq. (18) correspond to iterations of a random walk on the network. This process converges [41] to $S_i^{final} = k_i/2L$, where L is the number of links in the network. When $|\mathbf{S}|$ is not normalized to one, the final value for S_i is simply rescaled (and the interpretation of S_i as a probability is lost).

SOE thus lies outside of the Cybenko constraints. Its speed of convergence will still depend on the size of the gap of the resulting (asymmetric) matrix. However it has the undesirable feature that it converges to a nonuniform distribution.

We can fix this problem by noting that the converged values for SOE are proportional to k_i. Hence we define $\sigma_i = S_i/k_i$. We use the same rule as in SOE: Eq. (20) then becomes

$$\Delta\sigma_i = (1/k_i) \sum_{j=nn(i)} (\sigma_j - \sigma_i). \tag{21}$$

This is of the form of Eq. (14); but now the symmetry requirement on the α_{ij} is violated, since $\alpha_{ij} = 1/k_i$, while $\alpha_{ji} = 1/k_j$. The result is that this rule is not weight-conserving. It is still somewhat interesting, however. The rule (21) implies that

$$\sigma_i^{t+1} = (1/k_i) \sum_{j=nn(i)} \sigma_j^t \equiv \sigma_{av,i}^t. \tag{22}$$

In other words, this rule amounts to replacing the "signal" σ_i with its neighborhood average, in each time step. Hence we call this rule RNA (replacement by neighborhood average).

The RNA rule is of course just a rewriting of the SOE rule, in terms of the new variables σ_i. However this rewriting has meaning—for we can choose to let the movement of true load be guided by the rescaled σ_i values, rather than by the S_i, in Eq. (12). Also, even though the rule is not weight-conserving, we know that it converges (because SOE does); and we can find the final (uniform) value, as follows. We know that the final S_i values are $(\sum_j S_j^{(0)})(k_i/2L)$, where $S_j^{(0)}$ are the initial values. Then the final value for σ_i is just $\sigma_i^{final} = S_i^{final}/k_i = (\sum_j S_j^{(0)})/2L = (\sum_j k_j\sigma_j^{(0)})/2L$.

The SOE rule is not suitable for a signal diffusion law, because the load cannot converge to a uniform distribution if the signal does not. However it is not obvious that the signal movement must be weight-conserving. We want the signal to diffuse quickly, and to guide the load (which is assumed to move slowly), in a way similar to the two-phase approach, but one that runs both 'phases' in parallel and online. Weight conservation is not obviously necessary for the signal diffusion law. Hence we retain both FCDL and RNA as candidates for fast signal diffusion. Both depend solely on local information, and both can be quickly updated in response to topology changes.

We have discussed at some length the question of fast diffusion of signal, because the limit of fast diffusion is quite different in the discrete case from the same limit in the continuum (where nothing special happens). Now we have some candidates for fast diffusion, which amount to replacing rule (13); these candidates need to be evaluated under various conditions of topology and dynamics. It remains then to set up a "fairness criterion". That is, we want the load-moving ability of this chemotactic system to be calibrated, via the constant parameters c_2 and c_3 in equations (11) and (12), so that it is the same as that of plain diffusion. In other words, we want

$$\Delta\phi_{i,out}^{diff} \simeq \Delta\phi_{i,out}^{CT}. \tag{23}$$

We consider the FCDL rule (17) first. The LHS of our fairness criterion (Eq. (23)) is given in Eq.

(10). The RHS is

$$\begin{aligned}
\Delta\phi^{CT}_{i,out} &= \sum_{j=nn(i)} \Delta\phi_{i\to j} && (24)\\
&= \sum_{j=nn(i)} c_3 \cdot (S_i - S_j) && (25)\\
&= c_3 \cdot k_i \cdot S_i - c_3 \cdot \sum_{j=nn(i)} S_j && (26)\\
&= c_3 \cdot k_i \cdot (S_i - S_{av,i}), && (27)
\end{aligned}$$

where the last line defines the quantity $S_{av,i}$.

For FCDL, at any given time t, the signal at node i obeys

$$\begin{aligned}
S^t_i &= S^{emit}_i + (1 - \sum_j \alpha_{ij}) S^{t-1}_i + \sum_j \alpha_{ij} S^{t-1}_j && (28)\\
&= S^{emit}_i + M_{ii} \cdot S^{t-1}_i + \sum_j M_{ij} \cdot S^{t-1}_j. && (29)
\end{aligned}$$

Now we consider the extreme case that

$$k_i > k_j, \quad \forall j = nn(i); \tag{30}$$

that is, node i has higher degree than any of its neighbors, so that (for FCDL) all of its $\alpha_{ij} = 1/k_i$, and consequently $M_{ii} = 0$. Then we get

$$S^t_i = S^{emit}_i + (0) + S^{t-1}_{av,i}. \tag{31}$$

Hence, for this case,

$$S^t_i - S^t_{av,i} = S^{emit}_i - (S^t_{av,i} - S^{t-1}_{av,i}). \tag{32}$$

That is, a node which is a local maximum of node degree k_i has the two properties, under FCDL, that it sends out all its signal in one time step, and receives in return the average over its neighbors. Thus it is *locally* both SOE and RNA.

If we now consider the other extreme, namely that

$$k_i < k_j, \quad \forall j = nn(i), \tag{33}$$

we get that

$$S^t_i = S^{emit}_i + (1 - \sum_{j=nn(i)} \frac{1}{k_j}) S^{t-1}_i + \sum_{j=nn(i)} \frac{1}{k_j} S^{t-1}_j. \tag{34}$$

This differs from the result in (31) in that S^{t-1}_i gets more weight, and the S^{t-1}_j get less weight. For the purposes of finding a fairness criterion, we will treat these differences as roughly canceling, so that both extremes (and all cases in between) may be approximately represented by the results (31) and (32).

Now we make one further approximation. The two last terms in (32) may be of any sign, and have a time average of zero. Hence we will ignore them, in order to find a way of calibrating our FCDL chemotaxis rule against simple diffusion. This gives

$$\begin{aligned} S_i^t - S_{av,i}^t &\approx S_i^{emit} && (35)\\ &= c_2 \cdot (\phi_i - C_i); && (36) \end{aligned}$$

hence the RHS (27) of our fairness test becomes

$$\Delta\phi_{i,out}^{CT} \approx c_3 \cdot k_i \cdot c_2 \cdot (\phi_i - C_i). \quad (37)$$

This equation is readily compared with the LHS of our fairness test, see Eq. (10). We can then give the same brute force capacity to both plain diffusion and our chemotactic system by setting

$$c = c_3 \cdot c_2. \quad (38)$$

Now we can eliminate one unnecessary constant by setting

$$c_2 = 1; \quad (39)$$

that is, we measure signal in the same units as load and capacity. Our final result for enforcing fairness is then

$$c = c_3. \quad (40)$$

Finally we want a fairness test for the RNA rule, which is our second candidate for fast signal diffusion. We need the RHS of Eq. (27) (replacing S_i with σ_i everywhere). Including emission, we have

$$\sigma_i^t = \sigma_i^{emit} + \sigma_{av,i}^{t-1}, \quad (41)$$

and so

$$\sigma_i^t - \sigma_{av,i}^t = \sigma_i^{emit} - (\sigma_{av,i}^t - \sigma_{av,i}^{t-1}). \quad (42)$$

Here we make the same approximation as we did for Eq. (32), namely, we neglect the final two terms. Then we get (as before)

$$\begin{aligned} \sigma_i^t - \sigma_{av,i}^t &\approx \sigma_i^{emit} && (43)\\ &= c_2 \cdot (\phi_i - C_i), && (44) \end{aligned}$$

and hence we get Eqs. (37) and (38) again. Now we can in principle take a different value for c_2 for RNA; but we see no reason not to use the same as for FCDL, ie $c_2 = 1$. This gives us, finally, the result Eq. (40), so that RNA and FCDL can use the same values for their constants.

This is a simple result. We recall now the conditions under which our CT approach is expected to be useful: that the typical size of the load, compared to the typical bandwidth of the links, does not allow sending out most or all of the load in one time step—hence the need for signal (which can diffuse rapidly) to help the load as much as possible. Above, we expressed this condition as $c << c_4$. However, in our two candidate rules for fast signal diffusion, c_4 is not a constant. Instead, it is of order (in the case of FCDL), or exactly (in the case of RNA), $1/k_i$. Thus, if we want to simulate a coupled load/signal system, where the load moves much more

slowly than the signal, we want c to be less than, or much less than, the smallest "c_4" in the network, ie, we want

$$c < c_{4,min} = 1/k_{i,max}. \tag{45}$$

If there is a wide range of k_i values, then the simple inequality may suffice. For systems with little variation in k_i, it makes more sense to set c much less than $1/k_i$. Then, once c is determined, c_3 is determined as well.

3.3.3 Ants

Ant rules are typically known to perform aggregation algorithms. In this case, *inverse* rules are proposed, whereby the opposite effect—namely, load dissemination or load balancing—can be accomplished. The ants which roam about in the network pick up and drop the load following some rules. The rules are formalized by the set of simple algorithms.

First we give a rule for loaded ants.

Algorithm 3.3.1 *Loaded ants*
If Loaded
Move to node i, using Loaded Walk Rule
If $(\phi_i - C_i) < (\phi_i - C_i)_{av}$ and $(\phi_i - C_i) < 0$
Drop load $\Longrightarrow$ Unloaded

Here $(\phi_i - C_i)_{av} = (1/k_i)\sum_{j=nn(i)}(\phi_j - C_j)$ is simply the quantity $\phi_{av,i}$, corrected for the presence of capacities.

Now the rule for unloaded ants.

Algorithm 3.3.2 *Unloaded ants*
If Unloaded
Move to node i, using Unloaded Walk Rule
If $(\phi_i - C_i) > (\phi_i - C_i)_{av}$ and $(\phi_i - C_i) > 0$
Pick up load $\phi_{PU}(i) = c_{PU}(\phi_i - C_i) \Longrightarrow$ Loaded

These rules are similar in spirit and simplicity to those shown by Resnick to give aggregation behavior. Here the rules are designed to "anti-aggregate". We have not specified the rules for movement yet. The reason for this is that we will find it useful to define two different ant models: "blind" ants and "smart" ants. For blind ants we will be able to find and resolve the fairness test in a reasonable way. Then the smart ants' fairness test will be automatically resolved, since they have the same brute-force load-moving capacity, according to the above rules.

Now we tell how the ants move.

Algorithm 3.3.3 *Loaded Walk Rule*
Blind ants: random choice of neighbor
Smart ants: walk from j to i such that $(\phi_i - C_i) = \min_{k=nn(j)}(\phi_k - C_k)$

Algorithm 3.3.4 *Unloaded Walk Rule*
Blind ants: random choice of neighbor
Smart ants: walk from j to i such that $(\phi_i - C_i) = \max_{k=nn(j)}(\phi_k - C_k)$

In other words, blind ants move load randomly, while smart ants seek, if loaded, the least loaded neighbor, and, if unloaded, the most loaded neighbor. Note that all of our ant algorithms presuppose that every node possesses information about the loads and capacities of its neighbors. This information is needed for the Drop and Pick-up tests for all ants; so it is not unreasonable to let the same information guide the movement of the smart ants.

Now we need a fairness test. We want

$$\Delta\phi_{i,out}^{diff} = \Delta\phi_{i,out}^{ants}. \tag{46}$$

We will seek a way to enforce this criterion for blind ants.

We define $a_{u,i}$ to be the number of unloaded ants at i at a given time (time index suppressed); and let $p_{PU}(i)$ be the probability that an unloaded ant at i will pick up some load. Since the load test is deterministic, this probability is simply 0 or 1, depending on the load at i and its neighbors. However, we will use time-averaged quantities here, in order to simplify our fairness test.

With these definitions we have the RHS of Eq. (46) as

$$\Delta\phi_{i,out}^{ants} = a_{u,i} \cdot p_{PU}(i) \cdot \phi_{PU}(i). \tag{47}$$

Now we replace the product $a_{u,i} \cdot p_{PU}(i)$ by its time average $\overline{a_{u,i} \cdot p_{PU}(i)}$; and then, for blind ants, we note that the two terms in the product are uncorrelated—the random movement of the blind ants ensures that this is so. Then the product becomes $\overline{a_{u,i}} \cdot \overline{p_{PU}(i)}$, and we get

$$\Delta\phi_{i,out}^{ants} \approx \overline{a_{u,i}} \cdot \overline{p_{PU}(i)} \cdot \phi_{PU}(i). \tag{48}$$

The average number of unloaded ants at node i should be one half of the average number of ants at node i: $\overline{a_{u,i}} = \overline{a_i}/2$. Also, the two parts of the pickup test (the sign of the load-capacity difference, and how it compares to that difference averaged over neighbors) are uncorrelated. Hence $\overline{p_{PU}(i)}$ is the product of two probabilities, each of which is $1/2$.

Finally, we know the quantity $\overline{a_i}$. For ants whose movements are uncorrelated (blind ants), it is just the total number of ants A on the network, times the probability $p_{RW}(i)$ that one random walker will be at node i. This latter probability is known [41]: it is $p_{RW}(i) = k_i/(2L)$ where L is the total number of links on the network.

Now we put all of these pieces together into Eq. (48):

$$\Delta\phi_{i,out}^{ants} \approx \frac{Ak_i}{4L} \cdot \frac{1}{4} c_{PU} \cdot (\phi_i - C_i). \tag{49}$$

This expression is readily compared with Eq. (10). Equating the two gives

$$c = \frac{A}{16L} \cdot c_{PU}. \tag{50}$$

Here we find two parameters constrained by one. That is, only the product $A \cdot c_{PU}$ is determined by this result. It is interesting to note an implication of this result: namely, that if we wish to set the ants' "diffusion constant" c_{PU} equal to that (c) for our reference pure-diffusion system, we must have 16 ants on the network for each link of the network! No doubt having a high number of ants gives good performance, as it allows a more nuanced response to variations in (load $-$ capacity). The overhead required for so many ants may be greater than that for simple diffusion, however.

In any case we have some freedom of choice from our result (50). We will take $A = 16L$ and $c_{PU} = c$ as a good starting point for simulations.

Finally, we repeat our argument for smart ants: we claim that they have the same brute-force ability as long as they have the same pick-up test, and the same value for the product $A \cdot c_{PU}$. In fact, clearly the "most fair" comparison between blind and smart ants is obtained by giving both kinds of ants the same values of the individual parameters A and c_{PU}; and we will take this constraint as our fairness criterion for comparing blind and smart ants. Since we have also given the blind ants the same brute-force ability as plain diffusion, and since the CT system has been similarly constrained, we have found a set of conditions which give a (roughly) fair comparison for *four* different load-balancing systems: plain diffusion, chemotactic signaling, blind ants, and smart ants.

We address one last question regarding our ants here: does it make sense to call them "ants"?

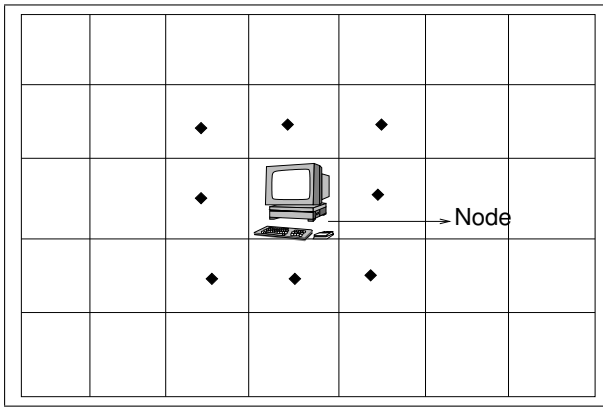
For instance, one could describe the plain-diffusion equations in words that invoke the idea of ants: we could let k_i ants be created at each node i at each instant, let them transport $c(\phi_i - C_i)$ to each neighbor j, and die there. Of course here we are knowingly trivializing the notion of ants! But perhaps there is some danger of unknowingly doing the same thing.

We have, in deliverable D03, listed several properties of real ants. Two in particular underlie many applications of "software ants": they are mobile, and they employ nondiffusive pheromone to mark paths. Here, we are only using the first property, their mobility. Nevertheless we argue that it is meaningful to refer to ants in this algorithm. Not only is it inspired by real ants; it is also qualitatively distinct from diffusion, in that the transport is discrete. That is, load is moved by countable, nonperishable units, and cannot be moved from i to j if there is no such unit (ant) at i. Furthermore, the "smart" ants choose where to move the load.

Our point is then that the algorithms described here are distinct from both plain diffusion and the chemotactic model; and they possess antlike characteristics in a nontrivial way.

3.4 Distributed Contents for p2p Networks Using Concepts from Natural Immune Systems

This work in progress, primarily developed by Dresden, is mainly aimed at developing an algorithm for searching p2p networks [18]. However, beyond the basic service of search, the algorithm takes care of arranging the contents in a fashion such that nodes possessing similar types of contents are clustered together. The fundamental ideas of the algorithm have been stolen from natural immune systems known for using robust, decentralized algorithms to kill diseases. As a result, the algorithm avoids query message flooding, instead of which employs concepts of proliferation and mutation which act as effective tool to produce high quality of



Note: The grid is toroid, i.e. the left side is joined with the right as well as the top is joined with the bottom.

Figure 8: A node in the p2p network and its neighborhood

search. The basic inspiration which we have derived from natural immune system in developing the algorithm is noted in D05. We are not repeating it here. The basic model parameters and the algorithms are next presented.

3.4.1 Simulation Model

This section is divided into two parts. In the first part, the framework used to model the p2p environment is described. In the second part, the algorithm denoted as *ImmuneSearch* responsible for producing distributed contents is elaborated.

Model Definition

The p2p network is modeled as a toroidal grid where each node in the grid is conceived as a member of the p2p network (Figure 8). Due to the grid structure, each node has a fixed number of eight neighbors, the neighbors are marked in Figure 8. Each node has two profiles - the *informational profile* and the *search profile*.

The *informational profile* (P_I) of the node is formed from the information which is shared by the node with other community members in the p2p network. The node's profile is created as a result of the analysis of data stored by the given node. The *search profile* (P_S) of the node is built from the informational interest of the users of a node. In general, this may differ from the information stored on the node. For simulation purpose, it is assumed that there are 1024 different categories of search. Moreover, it is assumed that these categories within themselves have some commonality. For example, if somebody searches for *pop* music, the interest may closely map with *rap* music. Hence, the search category *pop* and *rap* are conceived to be close to each other. Thus, the 1024 categories are represented by a 10 bit string, whereby proximity between each category is well defined. For simplicity, each node is assumed to have a single informational and search profile. The query message packet (M) is also a 10-bit binary token searching for a particular category. From now on we refer to category as token. Tokens which are close to each other in terms of Hamming distance are representing similar categories.

Similarities between a profile P and a query message packet (M), are measured in terms of the

number of bits that are identical. That is,

$$sim(P, M) = d - HD(P, M) \tag{51}$$

where HD is the *Hamming distance* between P and M and d is the length of the token. The 1024 tokens present in the network are distributed according to Zipf's law [57]. That is, if the most frequent token t_1 has occurred 1000 times, t_2 occurs 500 times, t_3 occurs 333 times and so on.

On the basis of this, the search algorithm *ImmuneSearch* is now elaborated.

ImmuneSearch

The *ImmuneSearch* is a distributed algorithm and is implemented individually and independently at each node. The search algorithm consists of two parts, the strategy followed by query message packets while moving through the network and topology evolution initiated by the search.

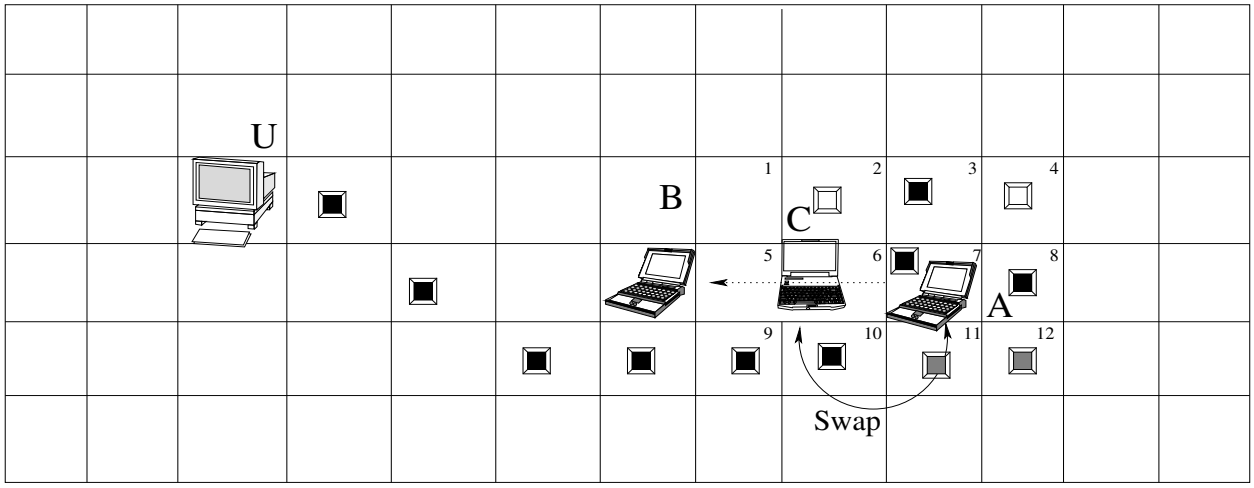
Movement: The search in our p2p network is initiated by the user node. The user (U in Figure 9) emanates message packets (which are derived from the *search profile* of the user) to its neighbors and the packets are thereby forwarded to the surroundings. The method of spreading the message packets to its neighborhood forms the basis of the algorithm. In the scheme, the spreading algorithm is inspired by the immune system. The message packets are conceived as *antibody*, whereas the profile it is searching is conceived as *antigen*. *For ease of understanding, the message packets are hereby referred to as antibody.*

In our system, the antibodies undergo random walk across the grid, but when they come across a matching antigen (an antigen is the information profile of any arbitrary node), that is, the similarity between the antibody and antigen is above a threshold, the antibody undergoes proliferation (as shown in position A of Figure 9), so as to find more and more nodes with similar information profile around the neighborhood. Besides proliferation, mutation is initiated which brings in diversity in the search and helps the user node to find a wide variety of search items. (The different colored message packets around A in Figure 9). The mutation results in finding new items which may not have been initiated exactly for search but which produce suggestive new results. This new suggestions can be helpful for the user. Moreover, the mutation also helps in bringing similar categories together. The mechanism is next presented in algorithmic form.

Each node executes the algorithm independently. The algorithm is initiated on a particular node as soon as an antibody (M) (query message packet) enters that node. Therefore, the algorithm named *Movement_p2p* is presented with respect to a particular node A.

Algorithm 3.4.1 Movement_p2p
Input: Antibody(M)
Output: Further Movement of the Antibody
If (sim(P_i,M) < Threshold)
Perform random walk on the grid.
else
Perform proliferation and mutation.

Proliferation and mutation initiate a differential movement around the neighbors of the nodes which are already found to be similar to the antibody. This implicitly points to the importance



Swap between C and A: C which resides in 6 (The nodes in the grid which are important to explain the search mechanism are marked from 1 to 12) has initially neighbors residing in {1, 2, 3, 5, 7 (= A), 9, 10, 11}; while A which resides in 7 has initially neighbors residing in {2, 3, 4, 6 (= C), 8, 10, 12}
After swap, A has neighbors {1, 2, 3, 5, 6 (= C), 9, 10, 11}, while C has neighbors {2, 3, 4, 7(= A), 8, 10, 11, 12}. That is, in swap, in effect, the neighbors are changed, not the physical position of the nodes.

Figure 9: The Search Mechanism

of topology evolution of the network which should ensure that nodes which have similar profiles come close to each other. This would naturally increase the effectivity of proliferation, because antibodies after undergoing proliferation will immediately begin to find nodes with similar *information profile*. This in turn ensures enhanced search output.

Topology Evolution: In the topology evolution scheme, the neighborhood of individual nodes are changed during search. Since each node can have eight neighbors, the neighborhood change follows the method illustrated in Figure 9. It is seen that the node has changed from A to B. During its movement from A to B, at each step it swaps its position that is, neighborhood information with intermediate nodes (as C in Figure 9).

The topology or the neighborhood configuration of a node (say A) is evolved based on the similarity between the node and the user node (say U) which has initiated a search. The similarity can be of two types;

- Similarity between P_I (*information profile*) of the node (A) and the antibody (M) initiated by the node (U).
- Similarity between search profile (P_S) of the node A and the antibody (M) initiated by the user node U.

The amount of movement of A towards the user node (U) is proportional to

- the similarity between them (A and U) either in terms of P_I or P_S.

- the distance between node U and node A.

The movement of nodes is also controlled by another important concept which is borrowed from natural immune systems - *aging*. The movement of a node gets restrained as it ages. It is assumed that the age of a node is dependent upon the number of times it encounters similar antibodies. That is, the longer it stays in the environment (p2p network), it is assumed that the node has found its correct position and the less is its response to any call for change in neighbors towards any user node U.

Moreover, if the search profile (P_S) of a node A matches with M, the movement of that node A towards user node U is initiated only if it is younger than U. If the node A has performed a search more times than U, it is assumed to be in a more stable position than U and consequently there is no further movement.

The aging concept lends stability to the system, thus a node entering the p2p network, after undergoing the initial changes in neighborhood finds the correct position. The algorithm for topology evolution is now elaborated. The *Topology Evolution* is executed on a node A, only when the similarity between antibody (M) and the profile (P) of A is above a threshold.

Algorithm 3.4.2 *Topology Evolution*
Output: Changed Position of A.
x_1 = *sim(P_I,M) /* P_I - Information Profile of A */*
x_2 = *age(A)*
x_3 = *dist(A, U)*
x_4 = *sim(P_S,M) /* P_S - Search Profile of A */*
x_5 = *age(A) - age(U)*
if ($x_1 >$ *Threshold*)
  mov(A) $\alpha \frac{x_1 \cdot x_3}{x_2}$
if ($x_4 >$ *Threshold and* $x_5 > 0$)
  mov(A) $\alpha \frac{x_4 \cdot x_3}{x_2}$

We are now in a position to present the main algorithm *ImmuneSearch* which is a combination of *Algorithms 3.4.1* & *3.4.2*. The ImmuneSearch is presented with respect to a particular node A. Each node applies the *ImmuneSearch* algorithm whenever it encounters an antibody.

Algorithm 3.4.3 *ImmuneSearch*
Input: Antibody (M)
Output: Movement
if (sim(P_I,M) > Threshold)
{
  Proliferate and mutate antibodies in neighborhood.
  Topology Evolution(A)
}
else
  Randomly forward the antibody M to any of the neighborhood.

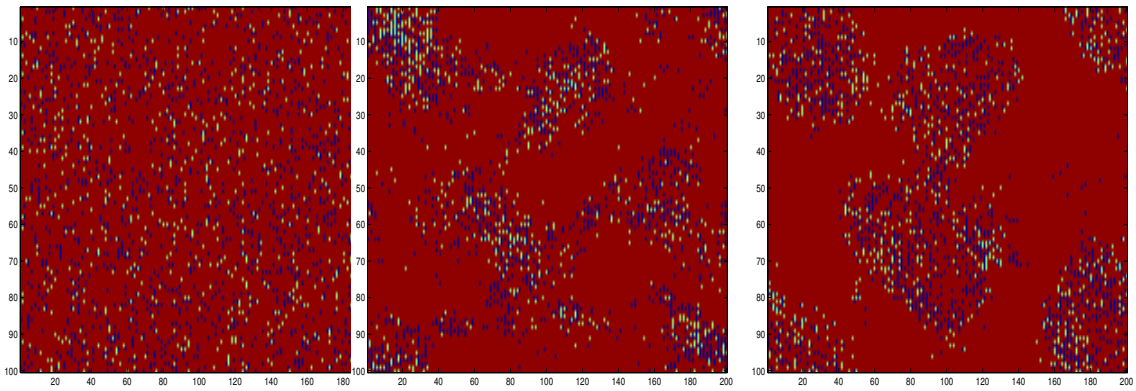


Figure 10: The clustering of similar nodes (nodes possessing the most frequent t oken) in a p2p network,the snapshots at generation no. 0, 3, 24 respectively

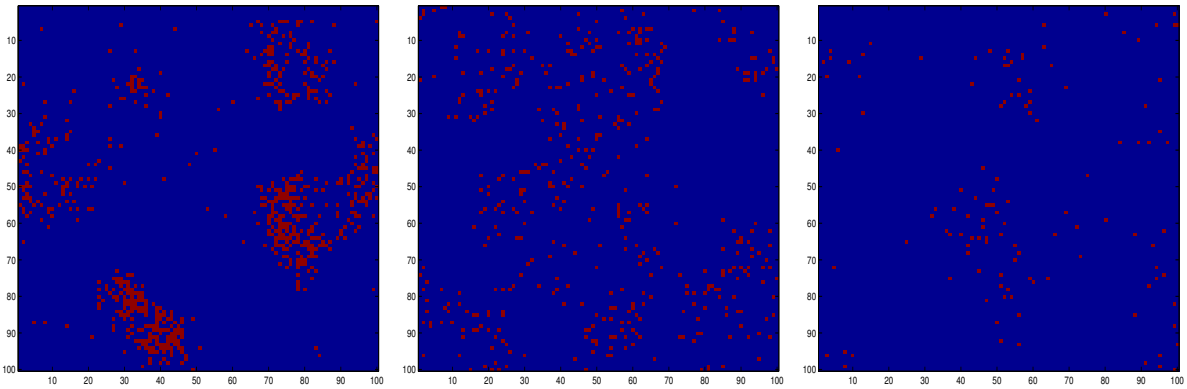


Figure 11: The clustering of similar nodes (nodes possessing 2^{nd}, 3^{rd} and 11^{th} most frequent tokens respectively) in a p2p network,the snapshots at generation no. 100

The initial simulation results which firmly establish the clustering effect of the categories and consequently enhanced search efficiency are discussed next.

3.4.2 Simulation Results

The experimental results point to the important feature of clustering of information, through the algorithm which in turn helps in performing a fast search scheme. The set-up parameters for the grid and the profile are elaborated next.

Experimental Setup: The experiment is performed on a 100 $\times$ 100 grid thus assuming that there are 10^4 computers participating in the network. Each search is initiated at a randomly chosen node and runs for 50 time steps. A generation consists of 100 searches. Each search event besides searching for the requisite items also arranges the peers in such a fashion that similar peers cluster together.

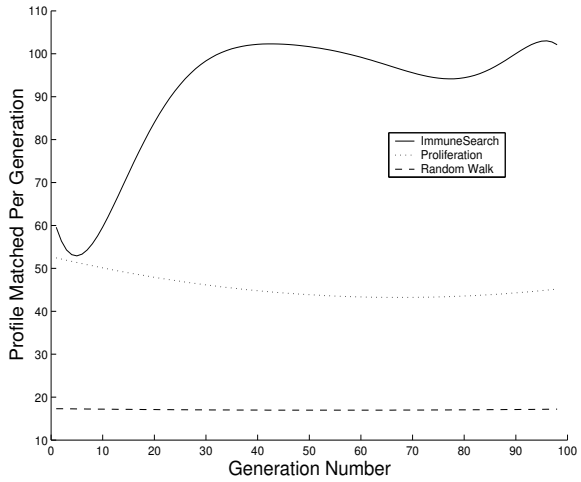


Figure 12: Efficiency of different techniques of search; The techniques are ImmuneSearch, Proliferation and RandomWalk respectively

Expt 1: Search in Steady Condition
The series of snapshots in Figure 10 & Figure 11 demonstrates the clustering effect in p2p networks (where the overlay network is a grid). These snapshots are result of experiments carried out with the assumption that no node/machine leaves the system. The figures show the 100 $\times$ 100 p2p grid. The x-axis in Figure 10 varies from 0 to 200, because each node displays its two profiles P_I and P_S. (The blue dots represent the search profile of a machine (P_S) while the yellow dots are the informational profile (P_I)). The three snapshots show the progressive nature of clustering of the nodes hosting the most frequently occurring token at generation number 0, 3, & 24 respectively. It is clear from the pictures that as time progresses, similar nodes tend to cluster within themselves. Thus, whenever the antibody finds one of the profiles, there is an avalanche effect in terms of proliferation and search output. These clusters provide a fantastic breeding ground for more and more antibodies, which progressively enhance the success rate of *ImmuneSearch*. Moreover, the nodes hosting less frequently occurring tokens also cluster within themselves. This is illustrated through Figure 11. The x-axis in Figure 11 varies from 0 to 100. Here we show only the informational profile; marked as red dots. The three snapshots show clusters of nodes possessing the 2^{nd}, 3^{rd} and 11^{th} frequent tokens. It is seen that these tokens, as a result of *ImmuneSearch*, also cluster within themselves.

The improvement of search through use of the ImmuneSearch algorithm is reported through Figure 12. Each search is initiated at a randomly chosen node and the number of search items (n_s) found within 50 time steps from the commencement of the search is calculated. The results in the figure show the average over 100 different searches whereby we obtain N_s where

$$N_s = \frac{\sum_{i=1}^{100} n_s}{100}$$

The value of N_s directly reflects the efficiency of the network. Consequently, in the graph (Figure 12), each generation number actually comprises of 100 searches. The search results are represented by the best fit polynomial method. This gives a clearer understanding of the trends of the results. The results of three types of experiments, the random movement, the

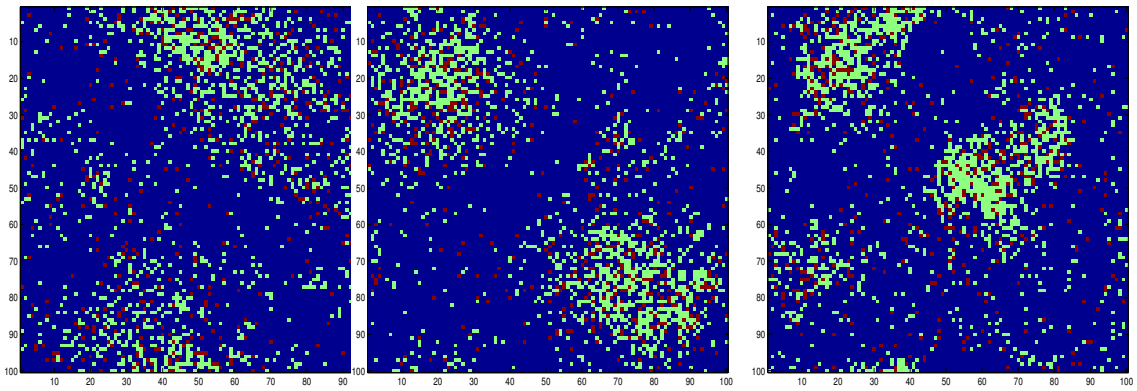


Figure 13: The clustering of similar computers in p2p Network in the snapshots at generation 100 for systems changing 5, 10, 20 percent peers respectively at each generation

proliferation and the *ImmuneSearch* are plotted. The x-axis of the graph shows the generation number while the y-axis represents the average number of search items (N_s) found in the last 100 searches. It is seen that the number of search items (N_s) found is higher in proliferation than in random search. However, as expected, N_s doesn't increase over time. In the case of *ImmuneSearch*, the number of search dramatically increases by almost 100% after initially starting in the same level as proliferation.

Expt 2: Search in Transient Conditions
The next experiment explores the behavior of the algorithm in face of constantly changing environment of $p2p$ networks. It is seen that even though nodes enter and leave the system at arbitrary pace, the *ImmuneSearch* algorithm organizes contents in a clustered fashion. The series of snapshots of Figure 13 show the robustness of the algorithm. The snapshots are produced as results of experiments where 5%, 10% and 20% of the population respectively are replenished after every generation. The snapshots, as in Figure 10 & *11*, show the position of the most frequently occurring tokens (marked in green) in the 100×100 grid: Besides the most frequently occurring token, also tokens which have Hamming distance 1 from this token are plotted (marked as red). The figure shows that even in face of such dynamic change, cluster formation is produced by the *ImmuneSearch* algorithm. Moreover, closely matching tokens also have a tendency to cluster together.

4 Outlook

This report presents the progress related to advanced services addressed by the BISON project. The advanced services are conceived as more complex services which cannot be addressed by a single input- output function. It is basically built based upon several basic services. A detailed state of the art has been presented.

Regarding the present work in progress, three different schemes are developed: two related to load balancing and one related to distributed contents. Some initial results of each of the respective schemes have been discussed. The schemes utilize the concept of epidemics, ants, chemotaxis and immune systems to develop these advanced services. Although some progress

is achieved, still the basic assumptions of the models are quite simplified. All the schemes need to be implemented in more realistic situations with higher degree of complexity to reap in the fruits of the initial basic theoretical work.

For example, the load balancing algorithm needs to be implemented in situations where each node has unequal capacity. The *ImmuneSearch* algorithm needs to be implemented on random topologies. Although implicitly the algorithms have considered storage and processing, bandwidth hasn't been taken into consideration for modeling till now. Moreover, none of the partners has already attempted to tackle problems related to *reliable on demand retrieval of data* in distributed storage which is an important major area. A lot of biological processes can be seen to employ different methodologies to circumvent unreliability in nature. A simple example is provided by crocodiles laying huge number of eggs to ensure that some of them ultimately give birth to new babies. We can learn from nature and build robust algorithms concerned with *reliable on demand retrieval of data*.

In the end, we would also like to put forward a new idea of advanced services. It concerns building up more sophisticated functions combining distributed processing and distributed storage. An important advanced function which can be thought of in the future is that, when we are distributing storage, that is we are warehousing a pool of data in some other machines, it may so happen that we would also like to process these data to extract information out of them. (A typical case is data mining). That means when we are sharing the load we have to balance load bearing two things in mind - the processing power and the storage capacity of the neighbors. In effect, this gives rise to a new interesting multi constraint load balancing problem.

References

[1] William Aiello, Baruch Awerbuch, Bruce M. Maggs, and Satish Rao. Approximate load balancing on dynamic and asynchronous networks. In *ACM Symposium on Theory of Computing*, pages 632–641, 1993.

[2] Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *Software Practice and Experience*, 2002.

[3] W. Bolosky, J. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proc. of Sigmetrics*, June 2000.

[4] John Byers, Jeffrey Considine, and Michael Mitzenmacher. Simple load balancing for distributed hash tables. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, February 2003.

[5] Y. Chen, J. Elder, A.Goldberg, A. Gottlieb, S. Sobti, and P. Yianilos. Prototype implementation of archival intermemory. In *Proc. of IEEE ICDE*, pages 485– 495, Feb 1996.

[6] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. Scan: a dynamic, scalable, and efficient content distribution network. In *Proceedings of the International Conference on Pervasive Computing (Pervasive 2002)*, 2002.

[7] Antonio Corradi, Letizia Leonardi, and Franco Zambonelli. Diffusive load-balancing policies for dynamic applications. *IEEE Concurrency*, 7(1):22–46, 1999.

[8] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. *Technical report, Computer Science Department, Stanford University, October 2002.*, 2002.

[9] George Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7:279–301, 1989.

[10] F. Dabek, M. Kaashoek, D. F. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. of ACM SOSP*, October 2001.

[11] Thomas Decker, Burkhard Monien, and Robert Preis. Towards optimal load balancing topologies. *Lecture Notes in Computer Science*, 1900:277–287, 2001.

[12] Ralf Diekmann, Andreas Frommer, and Burkhard Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7):789–812, 1999.

[13] Ralf Diekmann, S. Muthukrishnan, and Madhu V. Nayakkankuppam. Engineering diffusive load balancing algorithms using experiments. In G. Bilardi, A. Ferreira, R. Lueling, and J. Rolim, editors, *Solving Irregulary Structured Problems in Parallel (IRREGULAR '97)*, volume 1253, pages 111–122. Springer, 1997.

[14] Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Proceedings of the Workshop on Design Issues in Anonymity and Unobservability*, July 2001.

[15] P. Druschel and A. Rowstron. Storage management and caching in PAST, a largescale, persistent peer-to-peer storage utility. In *Proc. of ACM SOSP*, 2001.

[16] Fasttrack Home Page. http://www.fasttrack.nu.

[17] Freenet. http://freenet.sourceforge.net/.

[18] Niloy Ganguly, Geoffrey Carnright, and Andreas Deutsch. Designing of a fast search algorithm using concepts of natural immune system (*in preparation*).

[19] Johannes E. Gehrke, C. Greg Plaxton, and Rajmohan Rajaraman. Rapid convergence of a local load balancing algorithm for asynchronous rings. *Theoretical Computer Science*, 220(1):247–265, 1999.

[20] Bhaskar Ghosh, F. T. Leighton, Bruce M. Maggs, S. Muthukrishnan, C. Greg Plaxton, R. Rajaraman, Andréa W. Richa, Robert E. Tarjan, and David Zuckerman. Tight analyses of two local load balancing algorithms. In *STOC*, pages 548–558, 1995.

[21] Bhaskar Ghosh, S. Muthukrishnan, and Martin H. Schultz. First and second order diffusive methods for rapid, coarse, distributed load balancing (extended abstract). In *Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures*, pages 72–81. ACM Press, 1996.

[22] A. V. Goldberg and P. N. Yianilos. Towards an archival intermemory. In *Proceedings IEEE International Forum on Research and Technology Advances in Digital Libraries*, 1998.

[23] K. Hildrum, J. Kubiatowicz, S. Rao, and B. Zhao. Distributed object location in a dynamic network. In *Proceedings of Symposium on Parallel Algorithms and Architectures (SPAA)*, August 2002.

[24] Y. F. Hu and R. J. Blake. An optimal dynamic load balancing algorithm. Technical Report DL-P-95-011, 1995.

[25] Y. F. Hu and R. J. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25:417–444, 1999.

[26] S. Iyer, A. Rowstron, and P. Druschel. Squirrel: a decentralized peer-to-peer web cache. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing*, Monterey, CA, USA, July 2002.

[27] Márk Jelasity, Wojtek Kowalczyk, and Maarten van Steen. Newscast Computing. submitted for publication.

[28] Márk Jelasity and Alberto Montresor. Epidemic-Style Proactive Aggregation in Large Overlay Networks, 2004. To appear.

[29] M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, February 2003.

[30] Gene Kan. Gnutella. In Andy Oram, editor, *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, chapter 8. O'Reilly & Associates, March 2001.

[31] Kazaa. http://www.kazaa.com/.

[32] John Kubiatowicz et al. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural support for Programming Languages and Operating Systems*, Cambridge, MA, November 2000.

[33] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst*, 4(3):382– 401, July 1982.

[34] Alexander Loser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl, and Uwe Thaden. Semantic overlay clusters within super-peer networks. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing*, 2003.

[35] Q. Lv, P. Cao, E. Cohen, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th ACM International Conference on Supercomputing*, June 2002.

[36] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A scalable and dynamic emulation of the butterfly. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, 2002.

[37] Jim McCoy. Mojo nation, 2001.

[38] Dejan S. Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing. Technical Report HPL-2002-57, HP Labs, Palo Alto, 2002.

[39] M. Mitzenmacher, A. Richa, and R. Sitaraman. The power of two random choices: A survey of the techniques and results. *Handbook of Randomized Computing, P. Pardalos, S. Rajasekaran, and J. Rolim, Eds. Kluwer, 2000.*, 2000.

[40] Alberto Montresor, Hein Meling, and Ozalp Babaoglu. Messor: Load-balancing through a swarm of autonomous agents. In *Proceedings of the 1st International Workshop on Agents and Peer-to-Peer Computing*, 2002.

[41] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, UK, 1995.

[42] A. (Ed) Oram. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O Reilly Books, 2001.

[43] A. Rajagopala-Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured p2p systems. In *Second International Workshop on Peer-to-Peer Systems (IPTPS)*, February 2003.

[44] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the ACM SIGCOMM'01*, San Diego, CA, 2001.

[45] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the 18th International Conference on Distributed Systems Platforms*, Heidelberg, Germany, November 2001.

[46] Antony Rowstron, Anne-Marie Kermarrec, Miguel Castro, and Peter Druschel. Scribe: The Design of a Large-scale Event Notification Infrastructure. In Jon Crowcroft and Markus Hofmann, editors, *Networked Group Communication, Third International Workshop (NGC'2001)*, volume 2233 of *Lecture Notes in Computer Science*, pages 30–43, November 2001.

[47] G. Sakarian and H. Unger. Topology evolution in p2p distributed networks. In *Design, Analysis and Simulation of Distributed System*, pages 12–18, 2003.

[48] Peter Sanders. On the efficiency of nearest neighbor load balancing for random loads. In *Parcella 96, VII. International Workshop on Parallel Proccessing by Cellular Automata and Arrays*, pages 120–127, 1996.

[49] Tyron Stading, Petros Maniatis, and Mary Baker. Peer-to-peer caching schemes to address flash crowds. In *1st International Peer To Peer Systems Workshop (IPTPS 2002)*, Cambridge, MA, USA, March 2002.

[50] I. Stoica, D. Adkins, S. Ratsanamy, S. Shenker, S. Surana, and S. Zhuang. Internet indirection infrastructure. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, USA, June 2002.

[51] I Stoica, R Morris, D Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, San Diego, CA, 2001. ACM, ACM Press.

[52] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 175–186. ACM, ACM Press, 2003.

[53] P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmos. Towards high performance peer-to-peer content and resource sharing systems. In *Conference on Innovative Data Systems Research (CIDR)*, January 2003.

[54] Robbert van Renesse. The importance of aggregation. In André Schiper, Alex A. Shvartsman, Hakim Weatherspoon, and Ben Y. Zhao, editors, *Future Directions in Distributed Computing*, number 2584 in Lecture Notes in Computer Science, pages 87–92. Springer, 2003.

[55] C.-Z. Xu, Burkhard Monien, Reinhard Lüling, and F. C. M. Lau. Nearest neighbor algorithms for load balancing in parallel computers. *Concurrency: Practice and Experience*, 9(12):1351–1376, 1997.

[56] Cheng-Zhong Xu and Francis C. M. Lau. Optimal parameters for load balancing with the diffusion method in mesh networks. *Parallel Processing Letters*, 4(1-2):139–147, 1994.

[57] G. K. Zipf. *Psycho-Biology of Languages*. Houghton-Mifflin, 1935.