BISON
IST-2001-38923

Biology-Inspired techniques for Self Organization in dynamic Networks

Evaluation for Advanced Services in AHN, P2P Networks

Deliverable Number:	D10
Delivery Date:	January 2005
Classification:	Public
Contact Authors:	Andreas Deutsch, Niloy Ganguly, Geoffrey Canright, Tore Urnes, Márk Jelasity
Document Version:	Final (February 22, 2005)
Contract Start Date:	1 January 2003
Duration:	36 months
Project Coordinator:	Università di Bologna (Italy)
Partners:	Telenor ASA (Norway), Technische Universität Dresden (Germany), IDSIA (Switzerland)

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

Abstract

This document describes the evaluation of the models that have been developed so far in BISON in relationship to the management of advanced functions in dynamic networks. The models include an immune-inspired search and distributed content algorithm, chemotaxis-driven load balancing algorithms and adhesion-based structured topology management algorithms. This document is a follow-up of deliverable D08, which described the initial models for advanced services and reported the results of some preliminary tests. This deliverable reports the refinements of those schemes during the last year. Moreover, detailed experimental results comparing the performance with standard benchmark algorithms are presented.

Contents

1 Introduction

This deliverable deals with the evaluation of advanced functions. The earlier deliverables (D01, D05, D08) have defined the term "advanced". In BISON, we consider three functions as advanced. These functions are distributed processing, distributed storage and distributed content sharing and are mainly implemented on overlay networks. However, as in D04, here we do not distinguish between a simple search algorithm and distributed content management. This is because the chief objective of distributed content management is to improve the search efficiency. Furthermore, a simple search algorithm is always accompanied by more sophisticated techniques of content management. So, in the course of our research for two years, we have realized that it is unnecessary to treat these two functions separately.

This deliverable discusses (a) search and distributed content management using an immune system-based concept, (b) load balancing using the concept of chemotaxis.

In this deliverable we also include our preliminary results with a generic protocol for constructing a large set of overlay topologies. We call the protocol T-Man. This generic protocol can be used to develop advanced content management services like a *distributed hash table* (DHT). This line of research is new in the project and was not mentioned in D08, the original list of models to be evaluated.

2 Search and distributed content management using concepts from natural immune systems

This work, primarily developed by the Dresden team, is mainly aimed at developing an algorithm for searching p2p networks. The last year saw significant progress towards refining the algorithm. In the overview of the scheme which follows, we describe the algorithmic development in detail. The progressive development of the scheme has been published in the following conference proceedings [6,7,8,9,10].

The scheme can be divided into two major blocks: (a) devising an alternative movement strategy to random walk for fast propagation of messages, and (b) arranging the peers to form clusters of peers which host similar data so that the search operation can be speeded up. The preliminary algorithm has already been discussed in D08. In the second year of the BISON project, we have concentrated on improving part (a) of the algorithm. Hence, to avoid repetition from D08, the overview will mainly include the work developed for part (a) in detail. We do not elaborate the biological inspiration behind the development of the scheme, as that has already been discussed in D08.

The presentation of the work is divided into a number of subsections. The subsections are structured as follows.

1. **Description of the schemes:** As the scheme has already been discussed in D08, we report the refinements proposed during the last year.

2. **Benchmarks and expt. setup:** The benchmark algorithms are presented with which the performance of the proposed algorithms is determined. The metrics based upon which

corresponding comparisons are made are described. The experimental setup are also explained.

3. **Expt. results:** The experimental results are divided into two parts.
   - Experiments showing the comparison with benchmark algorithms with regards to traditional efficiency-related metrics.
   - Experiments done to test the 'nice properties' of the algorithms like adaptivity, scalability etc. as mentioned in D04 (Evaluation plan).
4. **Theoretical justification**, behind the superiority of the proposed schemes.

2.1 Overview

Decentralized peer to peer ($p2p$) networks like Gnutella are attractive for certain applications, because they require no centralized directories and no precise control over network topology or data placement. The greatest advantage is the robustness provided by them. However, since $p2p$ networks are unstructured, it is virtually impossible to perform a deterministic search operation. That means, a user node which wants to search for certain content cannot follow a deterministic routing protocol to reach the host. The two main alternative methods are either to flood the network with query messages, or to send k random walkers (message queries) into the network [19]. Flooding-based query algorithms, although seemingly fast, produce enormous amounts of traffic, and so can substantially slow down the system. k-random walkers don't generate enormous traffic, but are normally slow. With this perspective in background, we design a packet movement scheme based upon the immune system concept of opportunistic proliferation and mutation.

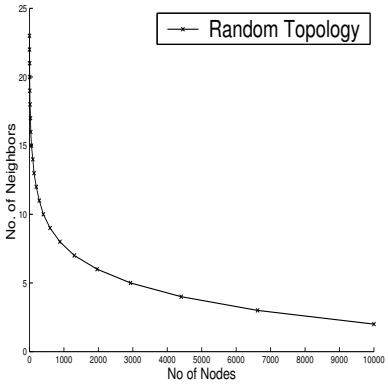
We first introduce the framework used to model the $p2p$ environment. While our simple models do not capture all aspects of reality, we hope they include the essential features needed to understand the fundamental qualitative differences between k-random walk and opportunistic proliferation algorithms. The different schemes are next elaborated. We finally describe the types of experiments conducted to compare between different schemes.

2.1.1 Model definition

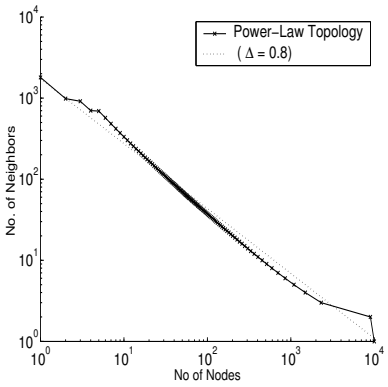
$P2p$ networks are networks formed through associations of computers, each providing equivalent services, eg. a search facility, to the network. Thus, each peer can be conceived as both client and server of a particular service [24]. To model a search service, we focus on the following important aspects of a $p2p$ system: $p2p$ network topology, query and data distribution, as well as the sequence of operation of the peers in the $p2p$ network.

A. Network topology

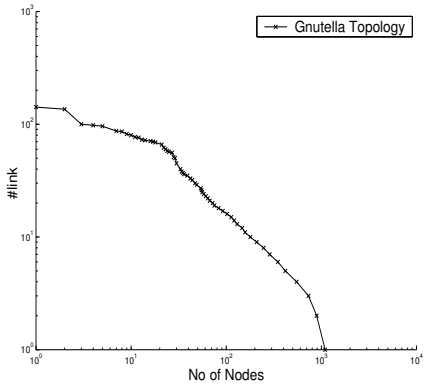
By network topology, we mean the graph formed by the $p2p$ overlay network; each $p2p$ member has a certain number of neighbors, and the set of neighbor connections forms the $p2p$ overlay network.



(a) Random graph, 10000 nodes, μ = 4



(b) Power-law graph, 10000 nodes, $\mu \approx 4$ and $\Delta \approx 0.8$



(c) Gnutella graph, 1095 nodes, $\mu \approx 7$

Figure 1: Distribution of node degrees in the three network topology graphs. Note that we use log scale for the power-law graph and Gnutella graphs, and linear scale for the random graph.

We use four different network topologies in our study. We have considered different parameters (eg. number of peers, mean indegree etc.) for each of the topologies. To explain the general features of the topologies, in *Fig. 1*, we plot the characteristics of a representative graph from random, power-law and Gnutella topology. The graph displays the number of neighbors each node has (y-axis) vs. the number of nodes having $\geq$ 'y' number of neighbors (x-axis).

- Uniform random graph : In the uniform random graph, the in-degree of each node follows a uniform distribution with mean μ. The graph is generated with the help of the topology generator BRITE [21].
- Power-law graph : The node degrees follow a power-law distribution; if one ranks all nodes from the most connected to the least connected, then the i^{th} most connected node has ω/i^{Δ} neighbors, where ω is a constant. The power-law random graph is generated

by topology generator Inet3.0 [15].

- Gnutella graph : The Gnutella network topology results were obtained from empirical measurements of the Gnutella network in 2001. Its node degree roughly follows a two-segment power-law distribution (which is true for most Gnutella graphs [16]).
- Two-dimensional grid: Two-dimensional grids having either 4 or 8 neighbors.

The data and query representation is discussed next.

B. Data and query representation

The peers host some data which are searched by other peers during a search operation. We have considered two types of data and query representation. The first one is simple, while the second scheme captures the essence of a file system in a much more realistic way. We name them *scheme simple* and *scheme realistic*, respectively.

B1. Scheme simple: In this case, we have assumed that there are 1024 different types of data present in the network, which can be represented by a 10-bit string. Hence, each peer hosts a 10-bit data string (say) P which we also refer to as its token. The query is also a 10-bit string.

B2. Scheme realistic: A file is represented as a list of keywords [18]. Hence the data distribution is represented in terms of keywords. It is assumed that there are 2000 different keywords in the system. Each node hosts some keywords. The number of keywords (not unique) in each node follows a Poisson distribution, with mean μ_{kw} = 1000. The data profile (D) of each node can therefore be represented as

$$D =< (\epsilon_1, n_1), (\epsilon_2, n_2), \cdots > \quad \& \quad n_1 + n_2 + \cdots = N$$

where ϵ_i are individual keywords, and n_i indicates their weights (number of times they are present in the node). N represents the total number of keywords.

The query consists of single or multiple keywords. The query (M) can be represented as

$$M =< m_1, m_2, \cdots >$$

where m_i represents each individual keyword. For 95% of the cases, the query length (say n) is ≤ 5, while it is between 6 and 10 in the remaining (5%) cases. In the 95% cases where the query length is ≤ 5, each length 1 to 5 is equiprobable. This is similar for the rest (5% case).

Distribution of keywords: Zipf's distribution [36] is chosen to distribute each of the unique keywords in the network in both schemes. In Zipf's distribution the frequency of occurrence of an event (here presence of a keyword) t, as a function of the rank r, where the rank is determined by the above frequency of occurrence, follows a power-law $t_i \propto \frac{1}{r^a}$. The ranking of keywords in terms of frequency is the same for both data and query distribution. For instance, the most popular query keyword is also the most popular data keyword.

The sequence of operation of the peers in $p2p$ network is next stated.

C. Operation sequence

The algorithms, which we have implemented (discussed in the next section), are distributed in nature; i.e., the peers perform the task independent of any centralized control. However, to assess the speed and efficiency of the algorithm, we have to ensure some sort of synchronous operation among the peers. In this context we introduce the concept of time, whereby it is assumed that in one time unit, all the nodes in the network execute the algorithm once. That is, if a peer has some message in its message queue, it will process one message within that time frame. We believe that, although approximate, this is a fair abstraction of the reality of $p2p$ networks, where each node is supposed to provide equivalent services. The sequence of operation of the peers during one time step is arbitrary.

We now describe the proliferation and random walk algorithms.

2.1.2 Algorithms

In this section, we explain two types of proliferation-mutation based search algorithms, as well as random walk based search algorithms. The important aspects of all these algorithms are that although random walk or proliferation-mutation is exhibited by the message packets, the algorithms are independently implemented by each node, and coordinated behavior of the nodes produces the required packet dynamics. All the algorithms can be expressed in terms of the same basic premise which is stated next.

Basic premise : The search in our $p2p$ network is initiated from the user peer. The user (U) emanates k ($k \geq 1$) message packets (M) to its neighbors - the packets are thereby forwarded to the surroundings. We next present the search initiation process in algorithmic form.

Algorithm 2.1.1 InitiateSearch(U)
Input : *Signal to initiate search.*
Form message packet (M) = $P_S(U)$
Flood k message packets (M) to the neighbors of the user peer.

The message packets travel through the network and when a node (say A) receives a message packet (M), it performs the following two functions.

- *Function 1* :- It checks the similarity between the incoming message M and the contents (D) of A. For simple and realistic datatype, the similarity measure differs.

  Similarity measure for simple data-type: Since both the incoming message M and D are tokens of 10 bits, a message is said to be similar to the content if M completely matches the content, that is,

  $$Sm = match(M, D) \Rightarrow Sm \; can \; have \; value \; either \; 0 \; or \; 1 \tag{1}$$

  Similarity measure for realistic datatype: Here the node checks whether any $\epsilon_i \in D$ of A is equal to any $m_i \in M$ (incoming message). The number of successful matches Sm is represented by the following equation.

$$Sm = \sum_{i=1}^{N} \sum_{j=1}^{M} (\delta_{m_j \cdot \epsilon_i}) \times n_i \quad (2)$$

where $\delta_{m_j \cdot \epsilon_i}$ = 1, if $m_j = \epsilon_i$, else 0. N is the total number of keywords present in the node, while M represents the length of the search query, n_i is the frequency of occurrence of the i^{th} keyword.

- *Function 2* :- The receiving node then forwards the content of the message packet in some *defined* manner to its neighbor(s).

In algorithmic form, we can represent the functions as *Reaction_p2p*:

Algorithm 2.1.2 Reaction_p2p(A)
Input : Message packet(M)
*If (Sm > 0) then {Report a successful match /*Function 1*/}*
Algorithm Message_Forward(A) / Function 2*/*

Each of the proliferation-mutation and random walk schemes defines algorithm *Message_Forward(A)* differently. Elaboration of the algorithms corresponding to each of the schemes follows.

A. Affinity-driven proliferation-mutation algorithm

Here we discuss two affinity-driven proliferation-mutation algorithms, the first one is what we term as simple while the second one is self-avoiding or restricted.

A1. Proliferation-mutation (PM) : In the proliferation-mutation scheme, the packets undergo proliferation at each node they visit. The proliferation is guided by a special function, whereby a message packet visiting a node proliferates to form N_{new} message packets which are thereby forwarded to the neighbors of the node. A randomly selected bit of each of these N_{new} messages has a probability β of getting mutated; β is the mutation probability of the system. Mutation is introduced into the system to increase the chance of message packets meeting similar items, which in turn helps in packet proliferation.

However, since mutation changes the content of the packet, it is assumed that the original information is also carried along with the packet. Hence, during the execution of the algorithm *Reaction_p2p*, comparison with D (Function 1) is carried out on the basis of the original message, while the input for *Algorithm Message_Forward(A)* (here Algorithm $PM(A)$) is the mutated packet.

Algorithm 2.1.3 PM(A)
Input : Message packet(M)
Produce N_{new} message packets(M)
Mutate one randomly selected bit of each of the N_{new} message packets with prob. β
Spread the N_{new} packets to N_{new} randomly selected neighbors of A

The function determining the value of 'N_{new}' ensures that N_{new} is $< n(A)$, where $n(A)$ is the number of neighbors of A and ≥ 1. [Note that if N_{new} = 1, proliferation/mutation behaves similarly to a random walk.]

A2. Restricted proliferation-mutation (RPM) : The restricted proliferation-mutation algorithm, similar to PM, produces N_{new} messages and mutates one bit of each of them with probability β. But these N_{new} messages are forwarded only if the node A has $\geq N_{new}$ free neighbors. By 'free', we mean that the respective neighbors haven't been previously visited by message M. If A has $\mathcal{Z}$ 'free' neighbors, where $\mathcal{Z} < N_{new}$, then only $\mathcal{Z}$ messages are forwarded, while the rest are destroyed. However, if $\mathcal{Z}$ = 0, then one message is forwarded to a randomly selected neighbor. The rationale behind the restricted movement is to minimize the amount of message wastage; we reason that two packets of message M visiting the same peer essentially means wastage of the second packet.

Algorithm 2.1.4 RPM(a)
Input : Message packet(M)
Produce N_{new} message packets (M)
Mutate one bit of each of the message packets with probability β
$\mathcal{Z}$ = No of 'free' neighbors
if ($\mathcal{Z} \geq N_{new}$)
    Spread the N_{new} packets in N_{new} randomly selected neighbors of A
else
        if ($\mathcal{Z} > 0$)
            Spread $\mathcal{Z}$ packets in $\mathcal{Z}$ free neighbors of A
            Discard the remaining (N_{new} - $\mathcal{Z}$) packets
        else
            Forward one message packet to a randomly selected neighbor of A
            Discard the remaining (N_{new} - 1) packets

We now elaborate the function which controls the amount of proliferation. Since the function of producing N_{new} packets depends on the similarity between the query and data present in the network, we have to define two different proliferation controlling functions for the two separate data types used - *simple* and *realistic*. However, in both cases, care is taken to avoid producing more than $\eta(A)$ number of messages, where $\eta(A)$ represents the number of neighbors of A.

Proliferation-controlling function for data of type simple: To formulate the proliferation-controlling function, we define the following expression p, where

$$p = e^{-HD} \times \frac{\rho}{\eta} \ .$$

Here HD is the Hamming distance (M,D); n represents the number of neighbors the particular node has; and ρ represents the proliferation constant (the same for all nodes). ρ is generally kept lower than the mean indegree of the underlying network. However, since in networks, the neighborhood distribution varies widely, if for some particular node (say B) $\rho > \eta(B)$, then for that node ρ is set to $\eta(B)$.

The number of message packets produced is given by the equation

$$N_{new} = 1 + p \times (\eta - 1) \tag{3}$$

Proliferation-controlling function for data of type realistic: The proliferation of message packets at any node A is heavily dependent on the similarity between the message packet (M)

and the data profile (D) of A. In this connection, we define the measure of similarity between the data profile (D) of the node and the message packet (M).

$$Sim = \frac{Sm}{N}$$

where the value of Sm is calculated through Eq. (2). [Note $Sm \leq$ N, so the value of $Sim \leq$ 1.] The number of packets N_{new} proliferated is defined on the basis of Sm in the following manner:

$$N_{new} = 1 + Sim \times (\eta - 1) \times \rho \tag{4}$$

where n represents the number of neighbors the particular node has, and ρ represents the proliferation constant, ($\rho \leq 1$). (ρ is set to 0.5 in all our experiments.) The above formula ensures that $1 < N_{new} \leq N$.

The random walk schemes are discussed next.

B. Random walk schemes

Here also there is the simple k-random walk algorithm and its restricted version. They are exactly similar to the proliferation-mutation algorithm except for N_{new} = 1. Besides these two versions, we also discuss high-degree restricted random walk, a special type of random walk suitable for power-law networks [1].

B1. k-random walk (RW) : In k-random walk, when a peer receives a message packet, after performing the task of comparison, as mentioned in *algorithm 2.1.2*, it forwards the packet to a randomly selected neighbor. The algorithm (RW) is quite straightforward and is defined as

Algorithm 2.1.5 RW(A)
Input : Message packet(M)
Send the packet M to a randomly chosen neighbor peer

The restricted random walk (RRW) algorithm which is similar to RPM (*algorithm 2.1.4*), is discussed next.

B2. Restricted random walk (RRW) : In RRW, instead of passing the message (M) to any random neighbor, we pass on the message to any randomly selected 'free' neighbor. However, if there is no 'free' neighbor, we then pass on the message to any randomly selected neighbor.

Algorithm 2.1.6 RRW(A)
Input : Message packet(M)
Send the packet M to a randomly chosen 'free' neighbor peer
If (no 'free' neighbor)
Send the packet M to a randomly chosen neighbor peer

The next algorithm is a special type of a random algorithm [1] to enhance the speed of simple random walk in a power-law network. The special type of random walk is termed as *high degree restricted random walk (HDRRW)*.

B3. High-degree restricted random walk (HDRRW) : Adamic et. al. in [1] showed that for a single random walker in a power-law network, *high-degree random walk* is better than simple random walk. To compare it with our proliferation-mutation scheme, we have simulated the algorithm with k-random walkers participating in the search. In the algorithm, a peer has special affinity to forward the message packet to the neighbors which have higher degree of connection. However, the algorithm also takes into consideration the restricted movement discussed in the previous section.

To balance these two trends, when sending a message packet, a peer checks the first $\mathcal{H}$ most high degree nodes to identify a 'free' node; if it doesn't find one, it randomly tries to identify any 'free' node another $\mathcal{L}$ number of times. If even then a 'free' node is not found, then the message is forwarded to a neighboring node. This forwarding scheme is also biased towards neighbors with high in-degree.

Algorithm 2.1.7 HDRRW(A)
Input : Message packet(M)
Find the 'free' neighbor with highest in-degree among the $\mathcal{H}$ most connected neighbors
If (no 'free' neighbor)
Randomly try $\mathcal{L}$ times to find a 'free' neighbor among the rest of the peers
If (still no 'free' neighbor found)
Chose a neighbor through a probability function $f(neigh)$;
where $f(neigh_1) > f(neigh_2)$ if $\eta(neigh_1) > \eta(neigh_2)$
Send the packet M to the chosen peer

We note the metrics based upon which the immune-based algorithms have been evaluated, and the benchmark algorithm with which it has been compared.

2.2 Benchmark algorithm and metrics

Our algorithm is compared with the k-random walk algorithm. The different versions of k-random walk algorithms (simple, restricted and high degree) have been discussed in D09. The efficiency of the proliferation-mutation-based algorithm (RPM, PM) is compared to that of these standard algorithms. Moreover, comparison is also made between RPM and PM - the two variants of proliferation-mutation algorithms which we have proposed.

In order to compare the efficiency of different algorithms, we have also to ensure fairness of 'power' among them. This idea is explained next.

2.2.1 Fairness in power

To ensure fair comparison among all the processes, we must ensure that each process (PM, RPM, RW, RRW, $HDRRW$) participates in the network with the same 'power'. The power can be considered in different ways. We categorize these one by one.

Fairness between a random walk and a proliferation-mutation algorithm :
To provide fairness in 'power' between a proliferation-mutation algorithm (say PM) and a random walk algorithm (say RW), we ensure that the total number of query packets used is roughly the same in all the cases. Query packets determine the cost of the search; too many packets cause network clogging, bringing down the efficiency of the system as a whole. It can be seen that the number of packets increases in the proliferation-mutation algorithms over the generations, while it remains constant in the case of random walk algorithms. Therefore the number of initial message packets (k_{RW}) in the random walk case is set so that the aggregate number of packets used by each individual algorithm is roughly the same:

$$k_{RW} = \frac{Tot\_Pack[PM]}{Tot\_Step[PM]}$$

where $Tot\_Pack[PM]$ is the total number of message packets used by PM to perform a particular experiment, while Tot_Step[PM] indicates the total number of time steps required to perform that task. The division is rounded to the nearest integer.

Fairness between two proliferation-mutation algorithms :
To ensure fairness in 'power' between two proliferation-mutation algorithms (say [PM & RPM]), we keep the proliferation constant ρ and the value of k (initial number of queries) the same for both processes. The value of k for the proliferation-mutation algorithm is generally set as

$$k = n(U),$$

where $n(U)$ is the in-degree of the initiator peer U.

Fairness between restricted and unrestricted algorithm:
Besides the cost of the message packets, during comparison between a restricted algorithm (say RRW) and a non-restricted algorithm (say PM), we also have to keep in mind that checking whether a node was visited previously involves a cost; this also should be taken into consideration when defining 'fairness'. Therefore, the composite cost[1] for a restricted algorithm can be defined as $C_{comp} = X + \gamma \cdot L$, where X is the average number of message packets, L is the number of neighbor lookups, while γ is the ratio of cost of lookup to cost of actually sending the message; γ normally ≤ 1. We consider γ either as 1 or 0 to discuss various possible scenarios.

The metrics are discussed next.

2.2.2 Metrics

In our work, we focus mainly on efficiency aspects of the algorithms, using the following simple metrics in our abstract $p2p$ networks. These metrics, though simple, reflect the fundamental properties of the algorithms.

- *Success rate:* The number of similar items found by the query messages within a given time period.

[1]Henceforth, cost or simple cost indicates the cost of message packets while composite cost always means total cost of messages and neighbor lookup.

- *Coverage rate:* The amount of time required by the messages to cover a percentage of the network.
- *Cost per search output:* The number of messages required to output a single hit. Cost can be either simple or composite depending upon whether we consider neighbor lookup cost.
- *Effectivity per message:* The number of search items produced by a single message. It is the inverse of *cost per search output*.
- *Bad visits:* The number of times the same message re-visits the same node. If a message packet visits the same node more than once, it amounts to wastage of that packet. So, a good design should minimize the amount of *bad visits*.
- *HotSpots:* The amount of messages processed by the busiest nodes. A good design should take care of the fact that the busiest nodes should not be overloaded which can in fact slow down the entire system.

Besides these efficiency-related metrics, experiments are also performed on a wide range of scenarios to test whether proliferation algorithms are scalable.

We now discuss briefly the experimental setup to measure the above metrics.

2.2.3 Experimental Setup

To explore the different properties of the algorithms, two major types of experiments have been performed on different topologies and with different initial conditions. The experiments are noted one by one.

COVERAGE: In this experiment, upon initiation of a search, the search operation is performed until the message packets cover the entire network. The experiment is repeated 500 times on randomly selected initial nodes. During the experiment, we collect different statistics at every 10% of coverage of the network. That is, we collect statistics at [20%, 30% $\cdots$ 90%, 100%] of coverage of the network[2].

TIME-STEP: In this experiment, upon initiation of a search, the search operation is performed for $\mathcal{N}$ (= 50) time steps. The number of search items (n_s) found within 50 time steps from the commencement of the search is calculated. The experiment is repeated for one generation, where one generation is defined as a sequence of 100 such searches. The search output (n_s) is averaged over one generation (100 different searches), whereby we obtain N_s, where

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

The value of N_s provides an indication of search efficiency.

The above mentioned two experiments have been performed for proliferation-mutation and random walk processes. The results derived from such experiments are noted next.

[2]Since the message forwarding algorithms (*Algo. 2.1.3 -2.1.7*) are non-deterministic in nature, the figures which plot the time time taken to cover [20%, 30% $\cdots$ 90%, 100%] of the network, show that message packets find it increasingly difficult to visit the last 10% of the network. This is true for all the different variants of message forwarding algorithms (PM, RPM, RW, RRW, $HDRRW$). For this reason, some figures explaining COVERAGE experiment only show results upto 90% coverage.

2.3 Experimental results

We have performed a large number of experiments with different topologies and different parameters. However, when presenting the results, we report only those results which help to understand the dynamics of the system. Hence, we are omitting many results, which just support or reinforce an already reported result.

The experimental results are organized in the following way. At first we report results derived from performing coverage experiments. Then we present the results from time-step experiments. At the end, we discuss various scalability issues.

2.3.1 Coverage

We report a series of results which help us to understand the behavior of the algorithms in different topologies. We consider both types of data representation, simple and realistic, to explain our results. Various other parameters, such as the proliferation constant and the mutation constant, are varied. The first four results are based on simple data representation, while the fifth result explains the effect of coverage, while considering realistic data types. The size of the tokens for simple data is assumed to be 10, that is, there are 1024 different data in the network. However, in case of the Gnutella graph (result - IV), we consider tokens of size 7. Our observation is that mutation helps in improving the efficiency of *network coverage* only in power-law networks; therefore, for all other types of topology, we have assumed the mutation constant β = 0. We have considered simple cost for all the cases except for the last where we have calculated composite cost (hence γ = 1). The different parameters used in the experiments are summarized in *Table 1*. Each of the results are noted one by one.

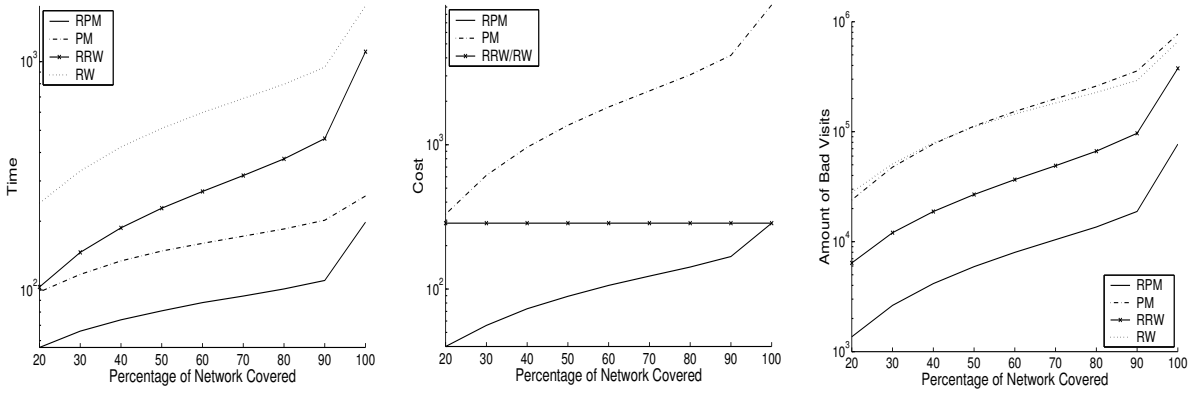
Result I : Coverage on grid

We consider a 100 $\times$ 100 two-dimensional grid (consisting of 10,000 nodes) and perform the *coverage* experiment; the results are plotted through *Fig. 2*.

Fig. 2(a) shows the network coverage rate of the RPM and PM algorithms at ρ = 3, as well as results for the RRW and RW algorithms. The graph plots the % of network covered in the x-axis, while the time taken to cover corresponding % of network is plotted on the y-axis (semilog scale). From the figure, it is seen that the time taken to cover the network is shortest in RPM followed by PM, RRW and RW.

Result	Data type	Topology	No. of nodes	Mean indegree	γ	β
1	simple	grid	10000	4	0	0
2	simple	random network	10000	≈ 4	0	0
3	simple	power-law network	10000	≈ 4	0	0.004,0.02,0
4	simple	Gnutella network	1095	≈ 7	0	0
2	realistic	random network	10000	≈ 4	1	0

Table 1: Summary of the parameters values used for performing the coverage experiment



a. Number of time steps required to cover the grid by RPM, PM, RRW & RW

b. Average number of message packets (cost) used by RPM, PM, RRW/RW

c. Number of bad visits by RPM, PM, RRW & RW

Figure 2: Graphs plotting average message use, network coverage time, and number of bad visits, for PM, RPM, RW and RRW

Fig. 2(b) plots the average number of packets used by the different algorithms (y-axis) vs. network coverage (x-axis). As expected, the number of packets increases in RPM/PM over the period of network coverage while it remains constant for random walk (RW/RRW) schemes. Therefore, while performing *coverage* experiments, the value of k for RRW/RW is set to the average number of message packets used by RPM to cover 100% of the network. Although the proliferation constant is the same for PM and RPM, from the figure it is seen that PM produces enormous amounts of messages, almost 10 times more than RPM. This happens because PM indiscriminately proliferates without checking whether the neighboring cells are already visited or not. The tendency is further detailed through the next figure (*Fig. 2(c)*).

Fig. 2(c) shows the number of *bad visits* (defined on pg. 15) by RPM, PM, RRW and RWs (y-axis) vs. network coverage (x-axis). It is seen that execution of both RW and PM results in a huge number of *bad visits*. Because the algorithms don't have any restriction, they have a tendency to visit the same node again and again. However, even though both RRW and RPM generally inherently try to avoid already visited nodes, we find that RPM can avoid such visits much more efficiently. (The number of bad visits by RRW is 10 times higher than RPM.)

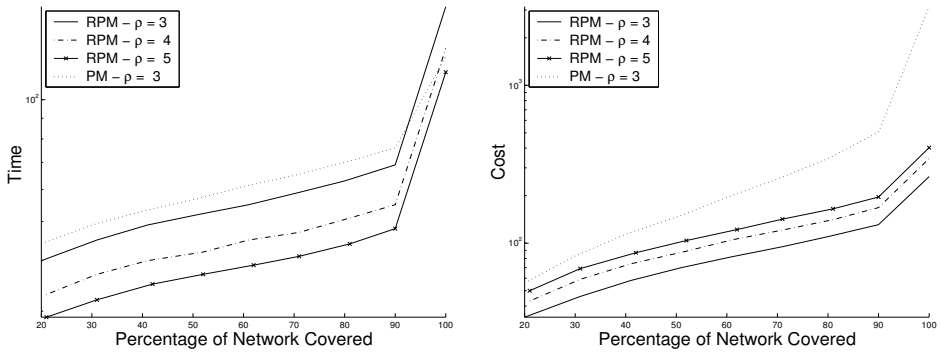
The next result shows that the superiority of RPM also holds for random networks.

Result II : Coverage on random networks

This experiment is performed on a random network of degree 4 with 10,000 nodes. The results reported below pertain to experiments performed for PM and RPM with different proliferation constants (ρ = 3, 4, 5), and for RRW. The major experimental observations are elaborated one by one.

Result IIa : Comparison between the PM and the RPM algorithm

Fig. 3(a) shows the network coverage rate of the PM algorithm at ρ = 3, and three different instances of the RPM algorithm at ρ = 3, 4, 5 respectively. The graph plots the % of network



a. Time taken to cover the network b. Cost incurred to cover the network

Figure 3: Graphs plotting (in semilog(y) scale) the cost and network coverage time of PM [ρ = 3] and RPM [ρ = 3, 4, 5], algorithms in random networks.

covered in the x-axis, while the time taken to cover the corresponding % of the network is plotted on the y-axis (semilog scale). It is seen that PM (ρ = 3) takes more time to cover up to 90% of the network than RPM with ρ = 3. Only while covering the last 10% does it overtake RPM. However, if we increase ρ for RPM, we see that even at 100%, RPM (ρ = 5) performs better than PM. An interesting observation to be noted is that RPM (ρ = 5) produces a smaller number of packets than PM (ρ = 3).

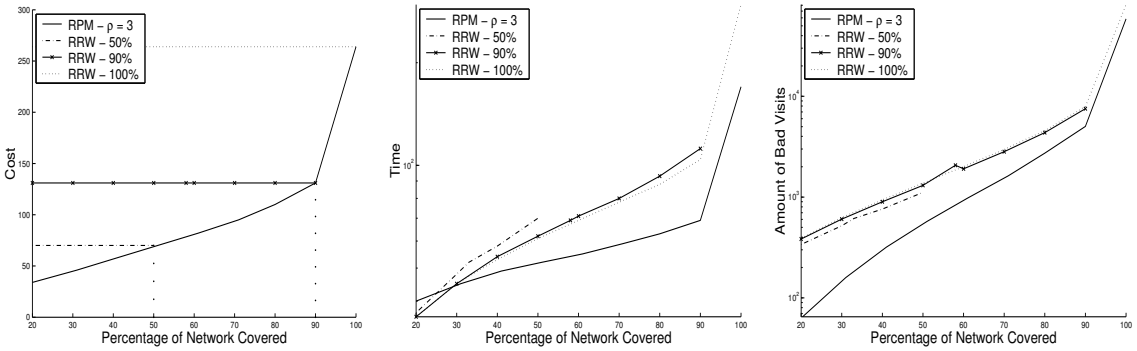
Fig. 3(b) plots the average number of message packets present in the network (also referred to as cost) in the y-axis (semilog scale) against the percentage of network coverage, for the four schemes. It is seen that the number of message packets produced in PM (ρ = 3) is about 10 times larger than RPM (ρ = 3, 4, 5). Similarly, in the case of random walks, it is found that RRW is much more efficient than RW (the experimental results are not reported here). We also find that RPM functions more efficiently in random networks than in the grid topology. (The time taken to cover 90% of the network in grid is 110 time units, while it is just 69 in random networks).

We next compare in detail the performance of RPM and RRW.

Result IIb : Comparison between the RPM and the RRW algorithm

The comparison between RPM and RRW is elaborated through the results highlighted in *Fig. 4*. In case of RRW, we perform three different sets of experiments by varying the initial number of k-random walkers. The three different experiments are termed as RRW(50%), RRW(90%), RRW(100%), respectively. The value of k for these experiments is set from collecting information about the average number of packets used by RPM(ρ = 3) to complete coverage of 50%, 90% and 100% of the network, respectively. *Fig. 4(a)* plots the average number of packets used by RPM, RRWs (y-axis) vs. network coverage (x-axis).

Fig. 4(b) plots the percentage of the network covered (x-axis) by the four processes RPM(ρ = 3), RRW(50%), RRW(90%), RRW(100%) against time step (y-axis, in semilog scale). It is seen that the time taken by RPM(ρ = 3) to cover the network is uniformly less than all the other processes beyond the 30% coverage. This is particularly significant because when we are in the range of 30%-40% network coverage, the number of message packets used by RPM is significantly lower than RRW algorithms (Refs. Fig. 4(a)). As expected, the time taken to



a. Average number of message packets used.

b. Network coverage time (semilog(y) scale).

c. Number of bad visits (semilog(y) scale).

Figure 4: Graphs highlighting (a) average number of message packets used (cost), (b) network coverage time, and (c) number of bad visits by RPM and RRWs ($RRW(100\%)$, $RRW(90\%)$ and $RRW(50\%)$) in a random network.

cover the network decreases progressively for RRW(100%), RRW(90%) and RRW(50%). This is because the three processes have an increasing number of message packets. However, as seen in the figure, the time does not decrease significantly even if we go on adding more messages. The RPM functions are better than RRWs because the packets here exhibit a much smaller probability to visit the same nodes again and again. *Fig. 4(c)* shows the number of *bad visits* (defined on page 15) performed by RPM and RRWs (y-axis) vs. network coverage (x-axis). It is seen that the tendency of RPM to visit the same node again and again is significantly lower.

We next highlight results from power-law graphs.

Result III : Coverage on power-law networks

This experiment is performed on the power-law graph with $\mu \approx 4$ with 10000 nodes. The experiment is carried out for RPM, RRW and $HDRRW$. In addition, the experiment is performed for RPM with different mutation probabilities. In the experiment, $HDRRW$ is especially taken into consideration. The major experimental observations are elaborated one by one.

Result IIIa : Comparison between power-law network and random network

Fig. 5, which is plotted in semilog scale, plots % of network coverage on the x-axis, with the time taken to cover the network space on the y-axis. The results for RPM and RRW are plotted for both random and power-law network, while for $HDRRW$, it is plotted only for the power-law network. It is seen that covering the power-law network is almost 10 times more difficult than the random network. The reason is that in power-law networks a few nodes possess a huge number of connections, while most of the nodes have very few connections. Hence, to reach a particular node (say x) from another node (say y) (both sparsely connected), the message has to pass through one of the more connected nodes, consequently creating an overload of messages on those more connected nodes and subsequently decreasing the speed of the total system. Similar to the results in random networks, the RPM works much better than RRW in power-law networks, too. The $HDRRW$ works better than simple RRW, but

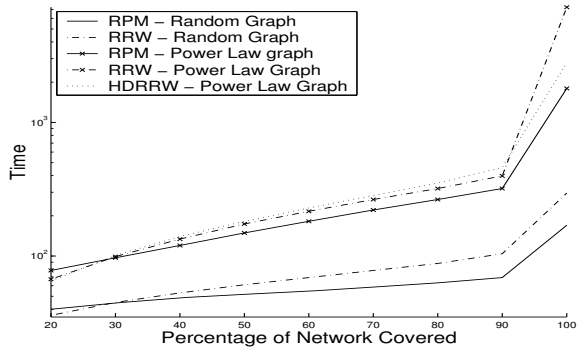


Figure 5: Graph showing network coverage time in random and power-law network by RPM, RRW and $HDRRW$.

only in the final stage of the coverage.

The next set of results shows that in power-law networks, mutating the proliferated packets indeed helps in improving the coverage rate.

Result IIIb : Effect of mutation on network coverage rate

Table 2 shows the time taken to cover the specified % of the network by PRM with β = 0.0, 0.004, 0.002, 0.01, 0.02 (*Column II - VI*). It is seen that mutation of the message packets speeds up the network coverage time. The RPM (0.004) covers the network fastest, exhibiting roughly 10% improvement over RPM without mutation. As we increase the mutation probabilities, we find that the coverage of the network initially is much faster, however RPM with lower mutation probability becomes faster after 90% coverage. One more important aspect which needs to be examined is the number of message packets produced due to mutation.

Fig. 6 plots the number of message packets produced at different mutation probabilities. It is seen that the number of message packets produced by RPM ($\beta > 0$) initially is almost the same as for RPM (β = 0). However the production increases in proportion to mutation probability as the % of network coverage increases. We find that the number of message packets produced by RPM (0.002) is 5% more than RPM (0), while for RPM (0.02) it is almost 25% more than

% coverage	RPM	RPM β = 0.002	RPM β = 0.004	RPM β = 0.01	RPM β = 0.02
20	78	68	72	67	57
30	97	86	90	84	77
40	120	108	113	105	98
50	149	135	141	129	123
60	182	166	173	157	152
70	221	204	210	190	183
80	265	249	255	232	221
90	320	302	313	283	272
100	1797	1590	1615	1612	1752

Table 2: Time taken to cover the network area by RPM algorithms with different β probabilities

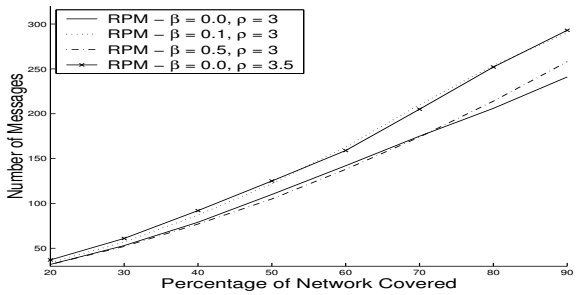


Figure 6: Graphs showing the cost regulation effect of RPM

RPM (0).

The initial increase in speed for processes with higher mutation probability can directly be attributed to the fact that they find a wide range of similar tokens. However, to cover the last 10%, more homogeneous packets are required. This is because to traverse the last 10%, messages have to travel though a lower number of designated paths, and the probability of finding those paths depends upon the similarity between the messages and a few nodes on those paths. A large mutation naturally weakens the homogeneous base, and thus the processes have more difficulty in finding those paths. To conclude, in order to effectively use the concept of mutation, regulation of mutation probability is absolutely imperative and the probability of mutation can be fixed in accordance to the type of search application one wants to perform. One more important aspect about mutation is that it produces the least severe hotspot which is discussed next.

Result IIIc : Hot spot formation by different processes

Fig. 7 plots the average number of messages processed by the most populated five nodes during execution of RPM, RPM (β = 0.002), RPM (β = 0.004), RPM (β = 0.01), RRW and $HDRRW$. RPM β = 0.004 produces the least severe hotspot (5910 messages). However, all the RPM produce more or less similar types of hotspot ($\approx$ 6000 messages). The hotspot produced by RRW is extremely severe (28000 messages), because it takes a lot more time to cover the network, thus loading the high degree nodes with a lot of messages. In $HDRRW$, we see that the hotspot is 50% more severe than that in RPMs in general.

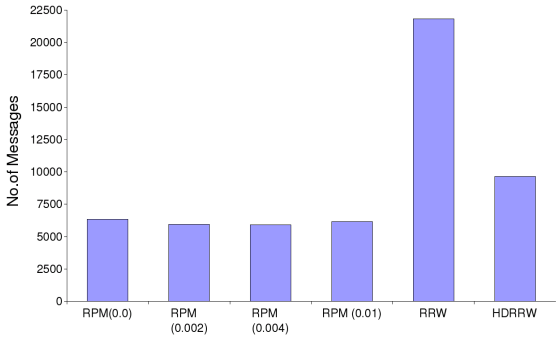


Figure 7: Bar chart showing the degree of hotspot formed by different techniques

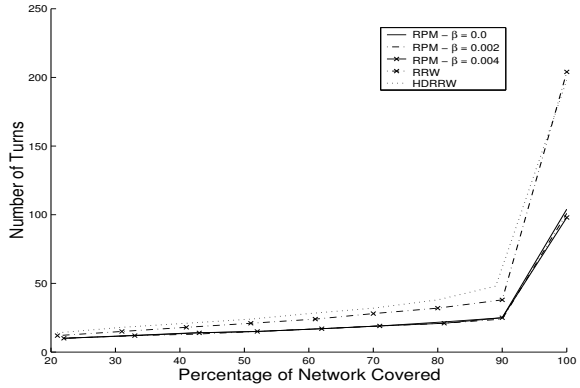


Figure 8: Graphs showing network coverage time of different algorithms in the Gnutella network.

Result IV : Coverage on a Gnutella graph

This experiment is performed on an empirically measured Gnutella Graph, consisting of 1095 nodes, with average node degree ≈ 7. It is assumed that there are 2^7 different tokens present in the graph. The experimental results are noted in *Fig. 8*. These results also show that RPM schemes are more effective than RRW schemes. Moreover, considering its power-law nature, it is harder to traverse the network than random graphs. A random graph which had 10,000 nodes could be traversed within 100 steps, whereas in Gnutella with a much higher in-degree, to traverse just 1095 nodes takes roughly the same number of steps. Moreover, in Gnutella, unlike power-law networks, we don't find the enhancement in performance of $HDRRW$ over RRW.

We now present the results obtained from experimenting with a more realistic data set.

Result V : Coverage on a random network - realistic data set

In this case, it is assumed that there are 2000 different keywords in the system. Each node hosts some keywords. The number of keywords (not unique) in each node follows a Poisson distribution with mean μ_{kw} = 1000. As we have seen in all the previous figures, due to the non-determinism of the algorithms, covering of the last 10% of the network takes unnecessary long time. This is true for all the different variants of message forwarding algorithms. Consequently, in the following results we avoid showing results from the last 10% as it only depicts effects arising from the finite size of the network.

Fig. 9(a) shows the network coverage rate of the different algorithms PM, RPM, RRW and RW. The graph plots the % of network covered on the x-axis, while the time taken to cover the corresponding % of network is plotted on the y-axis. It is seen that PM and RPM take almost identical time to cover the network. The time taken is, however, much less than that taken by RRW and RW respectively. The RRW is much more efficient than RW. We now assess the cost (both simple and composite) incurred by each algorithm to produce the above mentioned

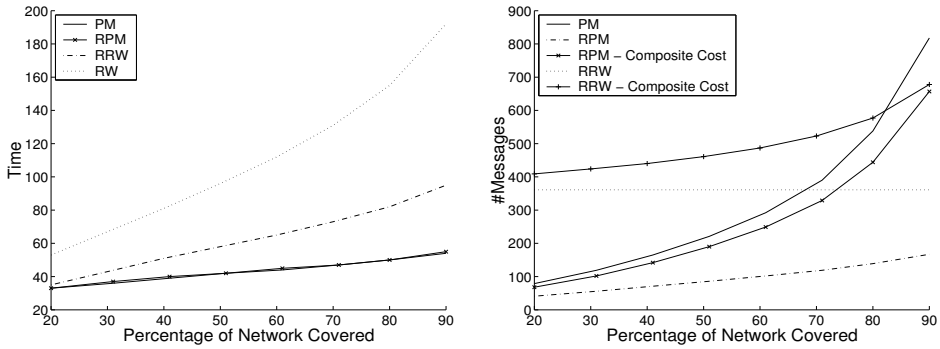
performances.

Fig. 9(b) plots the increase in the average number of message packets present in the network (also referred to as cost) on the y-axis network coverage for PM, RPM and RRW. For each RRW and RPM, we show two lines, one for simple and one for composite cost, respectively. Comparing, RPM and PM, we see that RPM uses a significantly smaller number of messages (about one-fifth) than PM and achieves the same performance. Even composite cost is significantly lower for RPM (657 for RPM and 818 for PM). To ensure the fairness criterion, RRW initially starts with the number of packets which RPM has used on the average to cover the entire network (361), so it stays constant throughout the experiment; however the composite cost steadily increases. It is the same as for RPM at the 90% coverage ratio. The number of messages RW uses (not plotted) is 1881. 1881 is the average composite cost incurred by RRW to cover the entire network.

We now summarize the conclusions which can be drawn from the experiments on coverage with different parameters.

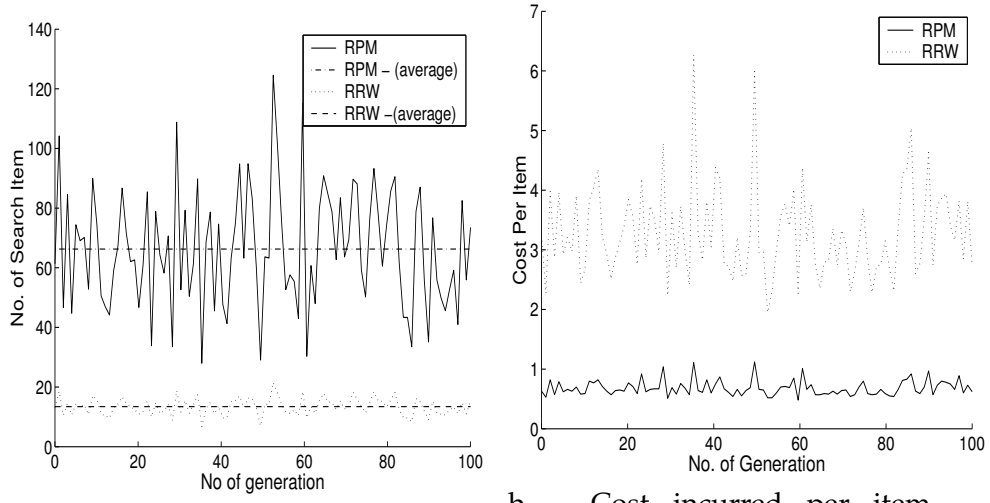
- The superiority of proliferation-mutation algorithms over random walk algorithms is independent of the underlying data distribution hosted by the $p2p$ network.
- Proliferation-mutation algorithms are fastest in random networks, followed by grid and power-law network.
- Mutation is helpful for power-law networks.
- We have noticed that the difference in performance between RPM and PM is significantly smaller for a realistic data set than for a simple data set. We believe this has to do with the difference in nature of the proliferation control function we have used for the two types of data sets; however, we need further investigation.

All the results point to the fact that the restricted version is better than the simple version. So, in our subsequent discussions, we drop PM and RW and concentrate on a comparison between RPM and RRW. The next experimental results highlight the search efficiency of RPM and RRW.



a. Time taken to cover the network b. Cost incurred to cover the network

Figure 9: Graphs plotting the cost and network coverage time of PM, RPM, RRW, RW algorithms in a random network. The proliferation constant considered here is ρ = 0.5.



a. Search efficiency of RPM (ρ = 3) and RRW

b. Cost incurred per item searched by RPM (ρ = 3) and RRW

Figure 10: Graphs showing search efficiency and cost incurred per item searched by RPM and RRW

2.3.2 Time-step

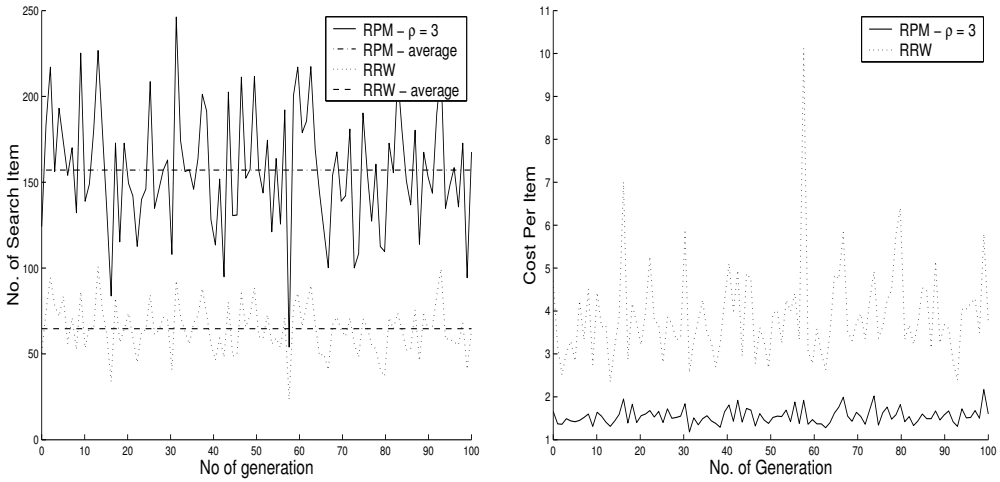
The parameters with which the time-step experiment is run have already been noted in *section 2.2.3*. We here highlight three results: (1) results derived on a grid; (2) results on random networks with simple data type; and (3) results derived on a random network with realistic data type.

Result I : Experiment on grid

To compare the search efficiency of RPM & RRW, we perform *expt: time-step* for RPM and RRW for 100 generations each. We consider a 100 $\times$ 100 two-dimensional grid (thus comprising 10,000 nodes) and perform the *time-step* experiment.

The graph of *Fig. 10(a)* displays the average value N_s against generation number for RPM and RRW. In this figure we see that the search results for both RPM and RRW show fluctuations. The fluctuations occur due to the difference in the availability of the searched items selected at each generation. However, we see that on the average, search efficiency of RPM is almost five times as high as that of RRW. (For RPM, the number of hits $\approx$ 65, while it is $\approx$ 13 for RRW.) The fluctuations in the results help us to understand an important aspect about cost/item which is discussed next.

Fig. 10(b) displays the cost/search item (the number of messages required to produce a search output) each scheme incurs to generate the performance of *Fig. 10(a)*. We see that the cost of RPM is hardly changing (it stays constant at around 0.7) even though the corresponding search output is differing widely, while in RRW there is significant fluctuation in terms of cost. This can be easily understood from the fact that RRW always starts with the same number of



(a) Search efficiency of RPM and RRW

(b) Cost incurred by RPM and RRW

Figure 11: Graphs showing search efficiency and cost regulation mechanism of RPM

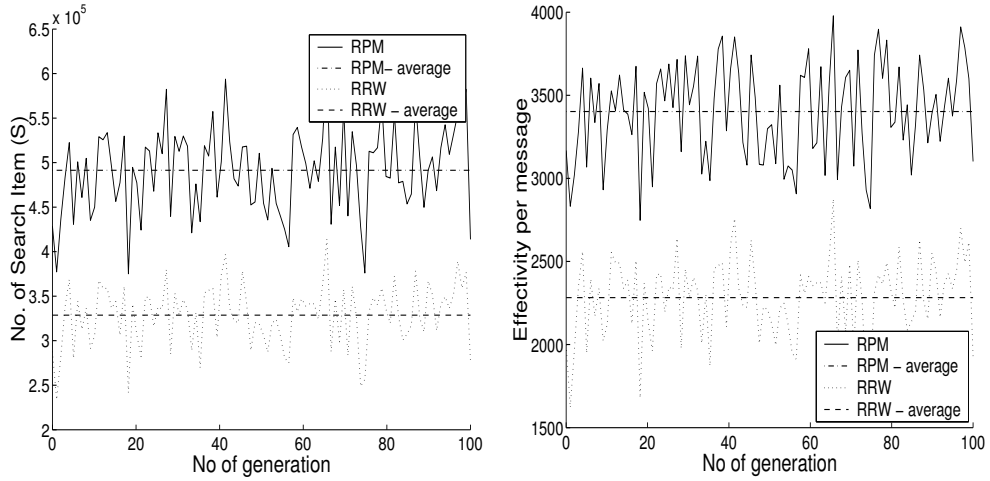
packets irrespective of the availability of the items. Therefore, when the search output is low, the cost shoots up sharply. In contrast, with RPM, the packets are not generated blindly, but are instead regulated by the availability of the searched item. Therefore, if a particular searched item is sparse in the network, RPM produces a lower number of packets (and vice versa).

Result II : Experiment on random network

To compare the search efficiency of RPM & RRW, we perform *expt: time step* for RPM and RRW for 100 generations each. The *time-step* experiment is performed on a random network of degree 4 with 10,000 nodes. The graph of *Fig. 11(a)* shows the average value N_s against generation number for RPM and RRW. 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. In this figure we see that the search results for both RPM and RRW show fluctuations. However, we see that on the average, search efficiency of RPM is almost three times as high as that of RRW. (For RPM, the number of hits ≈ 150, while it is ≈ 50 for RRW.) We notice that search efficiency is better than that obtained in grid topology (it has increased from 65 to 150). However, most noticeable is the improvement observed for RRW. It is seen that RRW has undergone 4-fold improvement while it is just 2.5 fold for RPM. However, we notice that the same cost regulation mechanism also operates here.

Fig. 11(b) displays the cost/search item (the number of messages required to produce a search output) each scheme incurs to generate the performance of *Fig. 11(a)*. We see that the cost of RPM is hardly changing (it stays constant at around 1.5) even though the corresponding search output is differing widely, while in RRW there is significant fluctuation in terms of cost.

Result III : Experiment on random network with realistic data set



a. Search efficiency of RPM and RRW

b. Cost incurred per item searched by RPM and RRW

Figure 12: Graphs showing (a). search efficiency, and (b). cost incurred per item searched by RPM and RRW in a random network.

To compare the search efficiency of RPM & RRW, we perform the *time-step* experiment on the random graph for RPM and RRW, each spanning over 100 generations. We use the same random graph as used to illustrate *result II*. The data distribution is same as that used to perform coverage experiments on realistic data sets (result V, section 2.3.1).

The graph of *Fig. 12(a)* shows the average value S against generation number for RPM and RRW. The x-axis of the graph shows the generation number while the y-axis represents the average number of search items (S) found in the last 100 searches. In this figure, we see that the search results for both RPM and RRW show fluctuations. However, we observe that on the average, search efficiency of RPM is almost 1.5 times as high as that of RRW. (For RPM, the number of hits $\approx 5 \times 10^5$, while it is $\approx 3.25 \times 10^5$ for RRW.)

The cost regulation which the previous two experiments have shown is also present here. *Fig. 12(b)* displays the effectivity per message (metrics defined in *sec. 2.2.2*) for each scheme. As expected, the effectivity of message packets for RPM is much higher than that for RRW (keeping in mind the fairness criterion we follow to generate experimental results). However, one more important point to be noted is that the standard deviation is particularly small in the case of RPM. For example, σ_E/μ_E (Standard deviation of effectivity/mean effectivity) is 0.1 for RRW, while it is just 0.08 for RPM.

To summarize:

- RPM is much more efficient than RRW, however, the efficiency varies with respect to topology as well as the data distribution.
- RPM has an excellent in-built cost regulation mechanism.

The next set of experiments investigates 'nice properties' of RPM. Since, RPM is a non-

deterministic algorithm as RRW, it is seen that the behavior of RPM is similar to RRW with respect to adaptivity, robustness etc (not shown here). The more interesting behavior is demonstrated when we test scalability of RPM. The experimental results are reported below.

2.4 Scalability test

In this section, we describe the different scalability tests performed. Three different types of scalability tests are chosen. Two are done on grid topology while the third is done on random topology. In the first type, we change the shape of the two-dimensional grid so far used, from the square grid (100 $\times$ 100) to a more rectangular shape (200 $\times$ 50, $\cdots$), and observe the impact it creates on the efficiency of RPM & RRW. In the second case, we increase the dimension of the grid (from 100 $\times$ 100 to 300 $\times$ 300 $\cdots$). The third experiment tests how efficiency of the different algorithms gets affected when the indegree of the random graph is increased. The first two experiments are performed with simple data-type while the third is performed on realistic data-type.

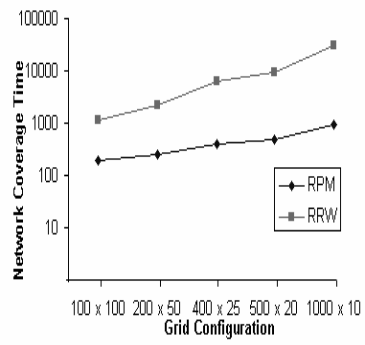
Result I : Shape and scalability

In this experiment, we consider different two-dimensional grids, each having 10,000 cells. However, the shape of each grid differs. The dimensions of such grids are respectively 100 $\times$ 100, 200 $\times$ 50, 400 $\times$ 25, 500 $\times$ 20, 1000 $\times$ 10. Testing the efficiency in all such configurations is particularly important because in real $p2p$ environments, we may not be able to develop exactly square connection among the peers and it is necessary for a proposed algorithm to perform well irrespective of the nature of the connection layout. We perform the *coverage* experiments on these differently shaped grid; the experimental results are illustrated in *Fig. 13(a)*.

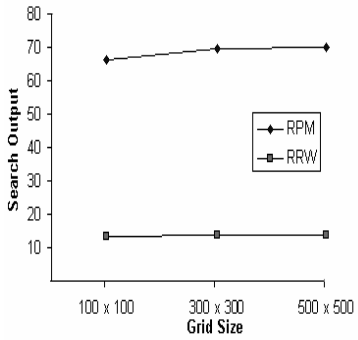
Fig. 13 (a) plots the network coverage time (y-axis) taken by RPM and RRW against different configurations (x-axis). For each, RPM and RRW, we plot the time taken to cover all the cells (nodes) of the grid. The figure shows that for both RPM and RRW, as the shape becomes more rectangular, the network coverage time increases. For example, RPM takes 198 time steps to cover the network when the grid is square-shaped, whereas it takes 1951 time steps when the grid has the rectangular dimension 1000 $\times$ 10. For the case of RRW, the respective figures rise from 1105 to 31025. Therefore, we see that RPM has a five-fold rise in network coverage time, while for RRW it increases by a factor of (around) 30. So, we can conclude that performance of RPM is much less sensitive to change in shape than RRW.

Result II : Size and scalability

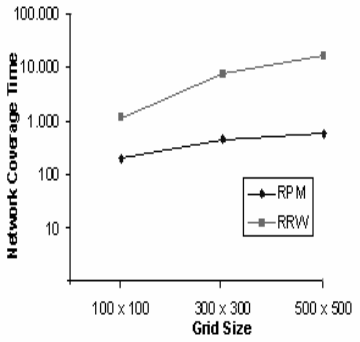
To measure the performance of RPM and RRW with respect to grids of different sizes, we consider grids of dimension 100 $\times$ 100, 300 $\times$ 300 and 500 $\times$ 500. We perform both *time-step* and *coverage* experiments with RPM and RRW on the three grids. In the case of the *time-step* experiment, a scalable algorithm should ensure that the number of search outputs does not deteriorate with the increase in dimension. *Fig. 13(b)* shows the result of the *time-step* experiment on the three different grids. It plots the average number of search output (y-axis) by RPM and RRW on different grid sizes (x-axis). The result shows that search efficiency for both



a. Network coverage time taken by RPM and RRW with different shapes of the grid, each grid having 10,000 cells.



b. Search efficiency of RPM and RRW at different grid sizes.



c. Network coverage time taken by RPM and RRW at different grid sizes.

Figure 13: Graphs illustrating scalability of RPM and RRW under different conditions

RW and RPM is independent of grid size. It is seen that the output of RRW virtually doesn't change while for RPM, in fact, it improves slightly when the grid size increases from 100 $\times$ 100 to 300 $\times$ 300. However, the output of RPM doesn't change when the grid size increases from 300 $\times$ 300 to 500 $\times$ 500. The reason for the initial enhancement of performance when the grid size increases from 100 $\times$ 100 to 300 $\times$ 300 is as follows. Since the grid in effect is a toroidal grid, the diameter of the 100 $\times$ 100 grid is 50. The *time-step* experiment is carried out for 50 time steps which results in some packets covering the entire length of the grid and colliding (visiting the same cells) with packets which have traveled to the edges from the opposite direction. This naturally decreases the output. When the grid is of size 300 $\times$ 300 or 500 $\times$ 500, there is no possibility of such collision. Note that the packet movement is much faster in RPM than RRW—since we don't see any improvement in RRW, we can conclude that the packets in RRW do not reach the edges within 50 time steps.

A more important test of scalability is shown in the next result, obtained by executing a *coverage* experiment, and depicted in *Fig. 13(c)*. *Fig. 13(c)* plots the time (y-axis) taken by RPM and RRW to cover 100% of the network against different configurations (x-axis). The coverage time in RPM increases from 198 (for 100 $\times$ 100) to 586 (for 500 $\times$ 500). The rate of increase is less than the logarithm of the rate of increase in the number of nodes. Any search algorithm in $p2p$ network with such low rate of increase is considered to be extremely scalable [4]. However, for RRW, the network coverage time increases from 1105 to around 16161, increasing almost linearly with the number of nodes.

Result III : Changing Indegree

In this experiment, we perform the *coverage* experiment on random graphs, each with 10,000 nodes, but with their mean indegree steadily increasing.

Fig. 14 plots the time taken to cover the network (y-axis) by RPM, PM, RRW and RW against different indegree configurations of the random graph (x-axis). The figure shows that as in-

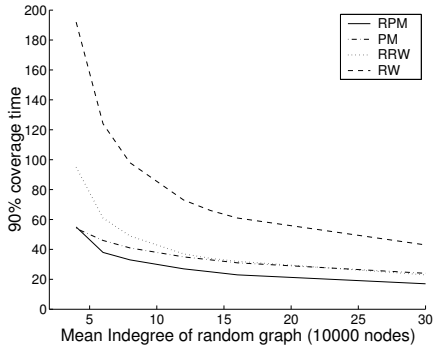


Figure 14: Graphs illustrating the effect on efficiency with varying indegree

degree increases each algorithm becomes faster. However, it has to be noted that the speed of random walk algorithms increases with a much faster rate. In fact, from around node indegree 12, RRW and PM take almost identical time. This happens because as the indegree of the random graph increases, in effect the dimension (d) of the graph also increases. That is, each node can reach each other node within a shorter time span. A random walk is particularly efficient at higher dimensions; this result points to that. However, maintaining high indegree in a $p2p$ environment typically is a problem [25]. So, at a lower indegree level, proliferation algorithms are much more effective than random walk algorithms.

We now summarize all the results obtained through our different experiments.

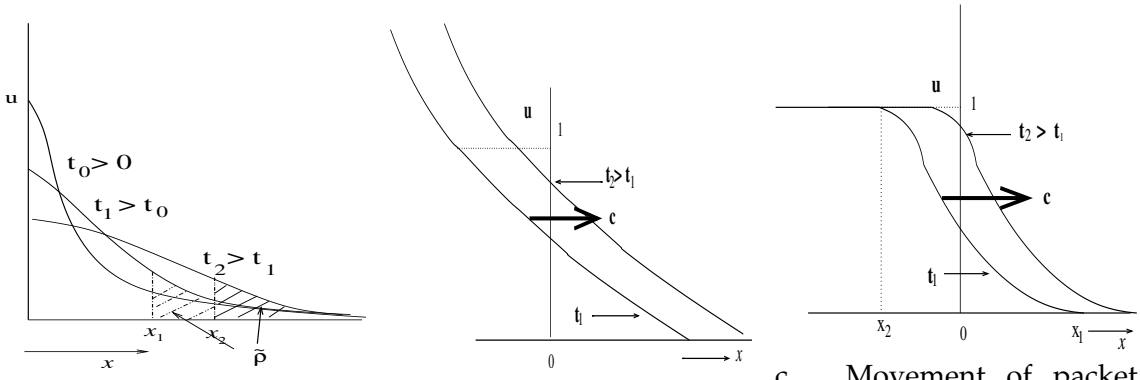
2.5 Summary of experimental results

The following is the summary of the results.

- RPM is more effective than PM.
- RPM is more effective than any random walk algorithm.
- RPM has an in-built cost regulatory mechanism.
- The search efficiency of RPM is higher than that of RRW.
- The RPM scales with respect to network size.
- RRW becomes more effective as the indegree of the graph increases.

2.6 Theoretical justification

In this section, we provide insights into the reasons behind the better performance of proliferation, as compared to random walk algorithms. We provide our explanation in the framework of a continuum model to derive the macroscopic behavior from knowledge about the individual microscopic behavior of the packets. Unlike the model described in the previous sections, for the sake of analysis we assume that the system is synchronous, in the sense that all nodes



a. Spreading of packet density u due to random walk (diffusion).

b. Movement of packet density u due to proliferation (reaction-diffusion 1).

c. Movement of packet density u due to restricted proliferation (reaction-diffusion 2).

Figure 15: Packet movement with different algorithms in a continuous system with radial coordinate x.

operate at the same time. The network is abstracted as a d-dimensional space where the fixed dimension d monotonously increases with the average node indegree.

We relate the random walk algorithms to a simple diffusive system where the diffusion starts from the origin. The proliferation algorithms can be conceived as reaction-diffusion systems [23], where besides diffusion of packets, new packets are continuously produced by the existing ones. Each of the two processes (diffusion and reaction-diffusion) spreads the message packets through the network. The insights which we will provide are based upon estimates of the *speed of packet spreading* in the radial direction.

Fig. 15 illustrates the basic features of the two processes. *Fig. 15(a)* refers to diffusion, *Fig. 15(b)* to a reaction-diffusion system with unrestricted proliferation, whereas *Fig. 15(c)* corresponds to a reaction-diffusion system with restricted proliferation. In all three graphs, we plot the density $u(x,t)$ of message packets used to conduct search versus the radial coordinate x. u can be conceived as a normalized measure of the number of packets k. u and k are not quantitatively related in this work, as that is not required to estimate the speed of packet spreading. Each of the three systems is studied in detail below. Here we discuss the figures one by one.

Fig. 15(a) shows three Gaussian curves at three different instances of time (t_0, t_1, t_2, where $t_0 < t_1 < t_2$). As time increases, a particular density of packets (say $\tilde{\rho}$) travels further away from the center. Moreover, since it follows a Gaussian distribution, at time $t \to 0$, at distance $x \to \infty$, there is some concentration of packets u, where $u \to 0$. However to cover all the nodes at distance x, a finite tangible concentration of packets $\tilde{\rho}$ should reach distance x. As can be seen from the figure, at time t_1, a concentration $\tilde{\rho}$ covers distance x_1 or more while at time t_2 the same concentration has covered $\geq x_2$ distance. Below we calculate the *speed of diffusive packet spreading* for this finite density $\tilde{\rho}$, which is approximated by a fixed but small value ρ.

On the other hand, in the case of reaction-diffusion systems (proliferation), the movement of packets follows a traveling front pattern with a uniform front profile. *Figs. 15(b),(c)* show such front profiles at time t_1 and t_2. For example, in *Fig. 15(c)*, we see that at time t_1, till distance x_2, the density u = 1, while beyond x_1, it is zero. The second curve at t_2 is just a uniform shift of

the first curve, hence characterized by a *front speed*. We now elaborate each of the processes one by one.

A. Random walk (diffusion): The random walk has traditionally been modeled as diffusion in continuum systems for which the diffusion equation reads [23]

$$\frac{du}{dt} = D \cdot \frac{d^2u}{dx^2} \tag{5}$$

where D is the diffusion coefficient. We are considering a situation where at time t = 0 all packets are concentrated at the origin from where they diffuse outwards. Hence, $u(x, t$ = 0) = $\delta(x$=0), where δ has non-zero value at 0 and 0 otherwise. Therefore, solving the differential equation with the given initial condition, we obtain

$$u = \frac{1}{(2 \cdot \sqrt{D \cdot \pi \cdot t})^d} \cdot e^{-\frac{x^2}{4 \cdot D \cdot t}} \tag{6}$$

where d is the dimension of the system.

We transform this equation in order to express the position x_ρ of an arbitrarily chosen fixed density $\rho \ll 1$ as a function of time.

$$x_\rho(t) = \sqrt{2 \cdot D \cdot d \cdot t \cdot \log \frac{1}{4 \cdot \rho^{2/d} \cdot D \cdot \pi \cdot t}} \tag{7}$$

The speed c of diffusive packet spreading for any fixed density ρ is obtained by differentiating $x_\rho(t)$ with respect to time.

$$c = \frac{2 \cdot D \cdot d}{2\sqrt{2 \cdot D \cdot d}} \cdot \frac{\log \frac{1}{4 \cdot \rho^{2/d} \cdot D \cdot \pi \cdot t} - 1}{\sqrt{t \cdot \log \frac{1}{4 \cdot \rho^{2/d} \cdot D \cdot \pi \cdot t}}} \tag{8}$$

For most of the time, the logarithm in the numerator is much larger than 1 and we can neglect the 1. Hence we obtain the simplified expression

$$c = \sqrt{\frac{D \cdot d}{2}} \cdot \sqrt{\frac{1}{t} \log \frac{1}{4 \cdot \rho^{2/d} \cdot D \cdot \pi \cdot t}} \ . \tag{9}$$

This result shows that packet spreading due to random walk becomes faster in higher dimensions (since $c \propto \sqrt{d}$), and is slowing down with time as $c \propto \sqrt{\frac{1}{t} \cdot \log \frac{1}{t}}$.

B. Proliferation (Reaction-Diffusion 1): The proliferation algorithm can be modeled as a system which is undergoing diffusion as well as gaining new packets as copies of existing ones

at the rate α at each time step. Therefore, the dynamics can be expressed by the following equation:

$$\frac{du}{dt} = D \cdot \frac{d^2u}{dx^2} + \alpha \cdot u \tag{10}$$

This equation resembles a variation of a well-studied reaction-diffusion equation, the Fisher equation [23]. We therefore utilize the standard result obtained for the *front speed* in a generalized Fisher equation with reaction term $f(u)$ [23]. The generalize Fisher equation is

$$\frac{du}{dt} = \frac{d^2u}{dx^2} + f(u) \ . \tag{11}$$

This equation has a uniformly moving front as solution; this front proceeds with the front speed

$$c = 2 \cdot [f'(u_1)]^{\frac{1}{2}}, \tag{12}$$

where $f'(u_1)$ denotes the derivative of $f(u)$ with respect to the packet density u at the density u_1. $u_1 = 0$ is the state of the system that has not yet been visited by the front.

Considering the equation (10), we can rescale it by

$$\begin{aligned} t^* &= \alpha \cdot t; \qquad & x^* &= x \cdot (\frac{\alpha}{D})^{\frac{1}{2}} \\ dt^* &= \alpha \cdot dt; \qquad & dx^{*^2} &= dx^2 \cdot \frac{\alpha}{D} \end{aligned} \tag{13}$$

Therefore, the equation (10) becomes

$$\alpha \cdot [\frac{du}{dt^*} = \frac{d^2u}{dx^{*^2}} + u] \ . \tag{14}$$

Hence $f'(u)$ = 1, which implies $f'(u_1)$ = 1. Therefore, the front speed c of the system is given by

$$c = \frac{\Delta x}{\Delta t} = \frac{\Delta x^* \cdot \sqrt{\frac{D}{\alpha}}}{\Delta t^* \cdot \frac{1}{\alpha}} = 2 \cdot \sqrt{\alpha \cdot D} \tag{15}$$

This result shows that the speed of packet spreading due to the proliferation algorithm is constant, *i.e.* independent of time. The speed depends on the proliferation rate α and diffusion constant D, but is independent of the dimension d. Hence, the behavior of the proliferation algorithm drastically differs from that of the random walk.

C. Restricted Proliferation (Reaction-Diffusion 2): In the model of restricted proliferation, the number of packets initially increases at a rate α; but the packet production rate is lowered as packets encounter more and more packets, *i.e.* as the density increases. We can write the proliferation function $f(u)$ using a logistic population growth model, as follows:

$$f(u) = \alpha \cdot u \cdot (1 - u) \ . \tag{16}$$

Therefore, the corresponding reaction-diffusion equation can be written as

$$\frac{du}{dt} = D \cdot \frac{d^2u}{dx^2} + \alpha \cdot u \cdot (1 - u) \ . \tag{17}$$

By using the same rescaling of space and time as above (13), we obtain

$$\alpha \cdot [\frac{du}{dt^*} = \frac{d^2u}{dx^{*^2}} + u(1 - u)] \tag{18}$$

Therefore, $f'(u)$ = 1 - 2 u and u_1 = 0. Hence $f'(u_1)$ = 1, and, following the same arguments as in case B, we find that $c = 2 \cdot \sqrt{\alpha \cdot D}$. This result implies the same speed of packet spreading and dependence on parameters as in the case of unrestricted proliferation.

To sum up, the above theoretical calculations of speeds of packet spreading due to different algorithms can explain the following observations of the coverage experiments and their dependence on the average indegree of the network (see *Fig. 14*).

1. Proliferation algorithms propagate packets faster through the network, as their speed is independent of time; whereas, for random walks, packet spreading slows down with time: $c \propto \sqrt{\frac{1}{t} \cdot \log \frac{1}{t}}$. This explains the differences in performance among PM, RPM and RW, RRW.

2. Restricted proliferation is as fast as the simple proliferation algorithm, in that, for both algorithms, the same speed $c = 2 \cdot \sqrt{\alpha \cdot D}$ was calculated. This result is consistent with our finding in *Fig. 9(a)* that the restricted proliferation scheme works as well as the proliferation scheme so that both curves PM, RPM coincide.

3. Random walk becomes faster as the effective dimension d of the network increases, *i.e.* the indegree increases. This explains the strong dependence of performance on the network indegree for both random walk algorithms in *Fig. 14*.

2.7 Summary and outlook

It is clear that in the past one year, we have done thorough extensive research to understand how proliferation can help in enhancing the speed and effectivity of the message packets. To

be honest, the idea we are using is extremely simple and we were surprised to find that the idea works well in unstructured networks. We initially thought that it will work only on semi-structured networks. This has directed us to undergo extensive experimentation as well as undertake various theoretical studies to find out the reasons behind the superiority.

The theoretical work is still going on (some of which we have not reported here, as it is not finished). We believe the next challenge is to define an efficient flooding mechanism, a mechanism which will not generate a huge number of packets like traditional flooding but will have comparable speed. Speaking more quantitatively, it can be shown that (multiple or single) random walk takes t^d times to cover a network (d is the dimension of the network), if flooding takes time t [34]. We are in the process of designing proliferation schemes which will take only t^2 time, yet will use a much lower number of message packets than flooding.

Moreover, we plan to carry forward the other important work of topology evolution to cluster nodes with similar contents. In D08, we have shown how similar contents can be clustered on grid topology and consequently, how search efficiency increases. In this year, we will extend this idea to other topologies especially power-law topology.

3 Efficient load balancing using chemotaxis

Load balancing is a function which may be used by other advanced functions, such as distributed storage (where load = stored data) and distributed processing (where load = cpu work to be done). In either case, it is often desirable to distribute the load over the participating nodes (peers) so that, at any given time, the load is evenly distributed. That is, the goal is to ensure that there is only small variation, over the network, in the local difference between a node's load and its capacity.

For a P2P system we want to solve the load balancing problem without resorting to any centralized control. Our chemotaxis-inspired approach to load balancing has been described in Deliverable D08. Here we simply repeat the basic working equations, for ease of reference.

Our starting point, or reference system, is "plain" diffusion. As we will see below, even "plain" diffusion can be viewed in many ways. However the basic idea of diffusion is that load moves away from a given node i more or less blindly (for example, *isotropically*). That is, in one unit of time, each node i sends out a certain fraction c of its load ϕ_i to each neighbor. In the "dumbest" form of diffusion, this fraction is independent of node, of neighbor, and of time:

$$\Delta\phi_{i\to j} = c \cdot (\phi_i - C_i) \tag{19}$$

for every node j which is a neighbor of i. Here C_i is the capacity at node i.

We will explore variations on this simple rule below. All such variations will still be considered as "plain" diffusion, as long as they do not use a chemotactic signal to guide the movement of load.

The point of chemotaxis is then to make the movement of load less blind, by giving the load a local signal which can guide it towards regions of low load. To do this, we let the load *emit*

signal at each time:

$$\Delta S_i^{emit} = c_2(\phi_i - C_i) \ . \tag{20}$$

As shown in D08, one can, without loss of generality, let the emission constant c_2 equal one.

Now we want the (chemotactic) signal S to diffuse away from the emitting load. In fact, we want it to move much faster than the load itself can move, so that it can carry information to more distant nodes:

$$\Delta S_{i \to j} = c_4 S_i \ . \tag{21}$$

The load can then respond to gradients of signal as follows:

$$\Delta \phi_{i \to j} = c_3(S_i - S_j) \ . \tag{22}$$

As shown in D08, a fair comparison of plain diffusion (with diffusion constant c) with our chemotactic approach requires that $c_3 = c$. The remaining constant c_4 need not in fact be a constant: we simply want our signal to move in a diffusive fashion, but much faster than the load moves. As known since at least the work of Cybenko [2], one cannot simply set the constant c_4 arbitrarily large, as this will result in instabilities. We have in fact studied "plain" diffusion rather thoroughly, in order to gain some feeling for what kinds of local diffusive rules we might use for the *signal* in the chemotactic system. That is, we seek rules giving fast diffusion for our signal, rules which thus lie close to but not beyond the threshold of instability. The rules we looked at do not always take the form of Eq. (21) (see below).

Finally, we note that, at the time of writing of this deliverable, our work with chemotaxis is not complete. We have explored plain (unaided) diffusive mechanisms on networks rather thoroughly. From this, we have chosen a small number of promising candidate approaches to guide the fast movement of signal. We have found that chemotactic diffusion according to Eq. (22) often suffers from instabilities. Nevertheless, at least one signal movement approach has shown that it is stable, and that it can give much faster (even for $c_3 = c$) convergence to a uniform fixed point than does ordinary diffusion. A closer inspection led us to believe that chemotaxis can be made much less prone to instabilities if nodes that contain less load respond to signal gradients in a more constrained fashion. We were able to find a simple constraint that rendered all our approaches stable, and that resulted in even faster convergence to a fixed point in many cases. We have a number of results for this constrained version of chemotaxis. However, though numerous chemotaxis simulations experiments have been performed, we have not yet completed the agenda of evaluations listed in Deliverable D04. As we report on evaluation results below, we will indicate areas in need of further investigation.

3.1 Overview of the algorithms

In the previous section, we have presented the basic ideas of plain diffusion, and of diffusion guided by chemotactic signal, in terms of simple equations. To implement these simple equations on a given network topology requires certain design choices to be made. Here we discuss those design choices.

3.1.1 "Plain" diffusion

There are two obvious undesirable features of the "most stupid" diffusion rule, Eq. (19). For one thing, whenever load at node i is less than capacity, it requires the node i to send out negative load. For another, it can cause the load itself (ϕ_i) to become negative. Each of these features is either unrealistic or meaningless; hence we have explored simple modifications of (19) which do not have these features. Each rule or set of rules that we have explored has been given a version number.

For example, our **version 1** of diffusion is simply Eq. (19). Version 1 can call for the sending of negative load; and it can lead to the load at a node becoming negative.

We also have examined a version (termed, for historical reasons, **version 0**) which makes a simple correction to version 1. That is, node i sends nothing if its load is less than its capacity; otherwise, it sends according to Eq. (19). This means that no node will send negative load. This constraint is expected to make the occurrence of negative load less likely also; but negative load is not ruled out by version 0.

Version 2 forbids any node, in any time step, from sending out more load than it has. That is, if node i has k_i neighbors, and if $c \cdot k_i \cdot (\phi_i - C_i) > \phi_i$, then Eq. (19) (and hence version 1) will send out more than ϕ_i from node i. (This may or may not make the new value of ϕ_i negative, depending on what arrives at i in the same time step.) To prevent this possibility, version 2 tests the above inequality, and if it is satisfied, requires node i simply to send amount ϕ_i/k_i to each neighbor—that is, to send out all the load that it has, but no more. This mechanism does not however prevent the sending of negative load from nodes with load less than capacity. Hence, version 2 also cannot completely rule out the occurrence of a negative load value at a node.

Versions 3 and 4 are not of current interest.

Version 5 is mathematically the same as version 1. However it avoids unnecessary load transfers, by finding the *net* transfer that is called for between nodes i and j by Eq. (19), for each link ij. That is, version 5 first finds the net difference $(\phi_i - C_i) - (\phi_j - C_j)$ across each link ij. [Note that, for all tests we have done so far, all nodes have the same capacity; hence this difference simply becomes $(\phi_i - \phi_j)$.] Then version 5 chooses the sending node to be that with the largest (most positive) difference between load and capacity; and this node sends only the net, positive, quantity $c \cdot |(\phi_i - C_i) - (\phi_j - C_j)|$. In practical terms, version 5 can make a big difference in bandwidth utilization (relative to version 1); hence we give it its own version number. Note that this version requires that each node knows the load (and capacity) at its neighbors at any time.

Version 6 allows each node to choose its own value for the diffusion constant c. This seemingly simple change requires some discussion. As noted in D08, and in Ref. [2], diffusion can be *anisotropic* (letting different links have different diffusion constants) without violating mass conservation; however it should be *symmetric* across each link. For the anisotropic case, if we denote the diffusion coefficient for i to send to j as α_{ij}, then symmetry requires that $\alpha_{ij} = \alpha_{ji}$. This in turn implies that nodes cannot choose their diffusion constants independently of their neighbors.

Version 6 works as follows. The system is given (by us, the simulators) a "default" diffusion constant c. However, any node i which discovers that $c > 1/k_i$ (where k_i is, again, its degree),

will adjust its own c value to be precisely $1/k_i$. Hence, two neighbors i and j who have both adjusted their values by this rule will have constants $1/k_i$ and $1/k_j$, respectively. This gives an (apparent) asymmetry, which can seemingly violate mass conservation.

In Deliverable D08 we proposed a rule like this one, with the further symmetry constraint that

$$\alpha_{ij} = \alpha_{ji} = \min(1/k_i, 1/k_j) \ . \tag{23}$$

We called this rule "fast Cybenko diffusion with a local criterion", or FCDL.

Version 6 differs from FCDL in two ways. First, there is a default c value for the nodes. Only when $c_{\text{default}} \geq 1/k_{min}$ (where k_{min} is the smallest node degree in the network) will *all* links ij be forced to choose between $1/k_i$ and $1/k_j$, as called for in FCDL. Secondly, version 6 enforces symmetry in a different way from that of Eq. (23). That is, version 6 (like version 5) first finds the net difference $(\phi_i - C_i) - (\phi_j - C_j)$ across each link ij. Then it chooses the sending node as in version 5. Finally, the sending node (say, i) sends $(1/k_i) \cdot |(\phi_i - C_i) - (\phi_j - C_j)|$ to its receiving neighbor j. This use of the same "instantaneous" c for both ends of a given link makes version 6 mass conserving. At the same time, version 6 has the chance to be even faster than FCDL. FCDL chooses a *static* value for each link; and, to be safe, it must choose the *lower* of the two possible values. Version 6, in contrast, can sometimes choose the larger of the two values.

In short: version 6, with $c_{\text{default}} > 1/k_{min}$, is like FCDL, with the additional advantage that it is adaptive over time, and so can choose the largest safe α_{ij} value at a given time for each link ij. Furthermore, the speed of version 6 can be continuously tuned: for maximal speed one can simply set $c_{\text{default}} = 1$; however smaller values of c_{default} make version 6 slower, while retaining its stability.

Version 7 is also version 5 plus a test. The protective rule is as follows. Each node i scans its neighbor list in random order. Each time a neighbor with a smaller load minus capacity difference than that of node i is found, we first check to see if sending out $c \cdot (\phi_i - C_i) - c \cdot (\phi_j - C_j)$ (the net load minus capacity difference) would result in node i getting negative load. If yes, then we don't send any load to the neighbor in question, but move on to the next neighbor instead. Notice that the net difference in load minus capacity that may be sent to neighbors typically will vary. If getting negative load prevented us from sending to one neighbor, we may still be able to send to another neighbor and maintain a positive load.

Version 8 is FCDL, unmodified, as given by Equation (23).

Version 9 is the algorithm termed RNA (replace with neighborhood average) in D08. We repeat the rule here for convenience:

$$S_i^{t+1} = (1/k_i) \sum_{j=nn(i)} S_j^t \equiv S_{av,i}^t \ . \tag{24}$$

This method is not suitable for load diffusion, as it is not weight conserving. However it is expected to be fast, and to converge to a uniform distribution. Hence it is a candidate for diffusion of signal (where weight conservation is not a requirement).

Version 10 is also only suitable for signal diffusion. The reason is that this method does not even attempt to maintain a strictly positive "load"—because we are here thinking of signal

diffusion. The algorithm proceeds as follows. During the first simulation cycle, each node subtracts its capacity from its load to get the starting value (ie, for signal). (Hence, when total load equals total capacity, method 10 converges towards zero signal on each node.) Now we want an algorithm that defines a sending node (say, i) for signal diffusion over link ij, so that the sending node can choose its own inverse degree ($1/k_i$) as the instantaneous diffusion "constant" over link ij.

We note here that, in the case of signal diffusion, the definition of sending node is less obvious than it is for diffusing (positive) load—since we expect signal values at a node, and signal values sent over a link, to be of either sign. Our generalization of the rule for plain diffusion is as follows: we define the sending node to be that node (of the two, i and j) which has its signal value farthest from zero—in other words, farthest from the signal value corresponding to the uniform fixed point distribution. In short: signal is sent to neighbors with a smaller absolute value of signal, i.e., neighbors j of node i that satisfy $|S_i| > |S_j|$. The amount of signal that is sent from i to such a neighbor then equals $(S_i - S_j)/k_i$.

Thus, version 10 is much like version 6. There are only two differences: (i) c_{default} is always large, ie, each sending node i always chooses $1/k_i$ for its diffusion constant; and (ii) the definition of sending node is modified to allow for the fact that the sent quantity (signal) can be negative.

3.1.2 Candidates for fast diffusion

We have investigated all of the above rules (versions 1–10) for diffusion. Our goal has been to pick out the rule or rules which give fast convergence without instability. We know from the work of Cybenko [2] that instabilities are found in the neighborhood of fast diffusion, and that the transition to instability occurs roughly when one or more nodes might be able to send out all of the load that they have in one time step. This qualitative rule has guided our design of versions 6–10.

We repeat our motivation here for clarity. We *assume* that the motion of load is forced to be slow by bandwidth constraints. This assumption is most likely to be reasonable when the load is large amounts of stored data. If we did not have this assumption, then we do not believe that chemotactic signaling can be of any help: one could simply find the fastest stable rule, and let the load itself diffuse according to that rule.

Instead (by assumption) the load can only move slowly, ie, with a very small c coefficient. Fairness then says that we must use the same value for the load's diffusion coefficient (as given by the constant c_3) when it is guided by signal gradients. However we can let the signal itself move fast, if we assume that a few bits can move very fast relative to (say) megabytes or gigabytes. Thus, for the signal, we are free to choose the fastest stable algorithm that we can find from the above list.

We have tested all of the versions 1–10 for plain diffusion. We have used a scale free network topology with 10,000 nodes for our tests, and a rather extreme starting condition, placing 10,000 units of load at a single node and none on the others. For each algorithm, we look for stability (ie, convergence to the uniform distribution from the given starting point) and for speed of convergence. We measure the latter by simply counting the number of cycles needed before all nodes' load values lie within 1% of the converged value (one). Capacities were set to one

($C_i = 1$) for all nodes and for all tests. Our tests were performed using the Peersim platform. For details of the implementation we refer to Deliverable D09.

Versions 1 and 5 performed as expected from the theory of Ref. [2]. That is, for small values of c, convergence was always obtained, albeit slowly. With increasing c, time to converge decreased steadily, but then began to increase sharply, until convergence was lost. This transition occurred around $c \approx 1/k_{max}$.

Versions 0, 3, and 4 did not significantly improve on the performance of our "dumbest" version (1 or 5); nor did they solve the problem of negative load values.

Based on the tests we have done so far, we find the fastest stable rules for diffusion to be versions 6, 8, and 10. Version 6 tends to be somewhat faster than version 8, as predicted by our argument above. Version 10 is only intended for signal, ie, for diffusion of quantities which may be of either sign. However, once one allows for this possibility, we find that version 10 is the fastest method of all the ones we have tested.

One key disadvantage of version 7 is that it was found not to converge in all cases. For example, when applied to a simple 10-node graph with a diffusion constant equal to 1/2, version 7 did not converge. (Interestingly, sometimes version 7 converged faster than version 6—for example, when using a diffusion constant equal to 1/5 for the same graph.)

Version 9 is not weight conserving. And so, in our tests, it has, as expected (see Deliverable D08) converged to the "wrong" fixed point—albeit rapidly. For example, for our extreme starting condition (see below) of all 10,000 units of load on the best connected node, the fixed point for RNA deviates most strongly from 0 (in this case, it converges to 571). For this reason, we have not yet carried out a thorough test of version 9 for signal, coupled to load. This remains to be done.

It is clear from the above that our analysis of methods for single-component diffusion (of load and/or signal) is extensive, and yet not complete. We have tested many different versions of one-component diffusion. However, we have only applied a simple convergence criterion for the one-component runs; and we have only tested two topologies, with a single initial distribution.

These tests gave us enough guidance to carry out the two-component tests described in the next section. These tests were done using version 6 and 10 for signal, since they are our current best candidates for fast and stable diffusion. However, we are not yet prepared to write off the possibility that some other method may prove to give even better performance than version 6 or 10, when coupled to load, for some or even most topologies and initial conditions.

3.1.3 Diffusion via chemotaxis

With the above results in hand, we are ready to implement diffusion of load, coupled to and guided by a rapidly diffusing signal which is emitted by the load itself. Our reference (unaided) system, for comparison, simply uses plain dumb diffusion, allowing the load to follow version 5. Negative load values are avoided in the reference system by simply letting c be small. Typically we chose c to be less than $1/k_{max}$, where k_{max} is the largest node degree in the network. This is of course a global criterion; but it satisfies our two purposes, namely, (i) to have slow diffusion, and (ii) to be able to compare slow unaided diffusion to slow diffusion that is guided

by signal. We implement the latter by setting $c_3 = c$, letting load by guided by Equation (22), and by using version 6 or 10 for the signal diffusion.

We note that we only send net loads in the signal-aided runs. That is, we implement diffusion according to Eq. (22) in a way that is strictly analogous to version 5 for plain diffusion: we find the net load to be sent over each link ij, and transfer that amount. In this case it is irrelevant which node "sends", since we use a network-wide common c_3 value.

Early experiments showed that when diffusion of load responds to signal according to Eq. (22), instabilities often resulted. A closer inspection of those early results, along with insights offered by the work of Cybenko [2], led us to believe that chemotaxis can be made less prone to instabilities if nodes that contain less load are more constrained in their response to signal gradients. We therefore augmented Eq. (22) with the following two constraints. First, only nodes with more load than capacity are allowed to send to neighbors. Secondly, the total load sent must be less than or equal to the difference between load and capacity of the sending node. The effect of these constraints is that once a node receives more load than capacity, it will maintain a load of at least capacity.

3.2 Evaluation of the algorithms

3.2.1 Conditions for the tests

We have so far performed a limited series of tests of the chemotaxis method. These tests were confined to two topologies. The main tests were done using a scale free topology with 10,000 nodes. Generation of this topology is described in D09. Some diagnostic tests were also performed on a much smaller test graph (with 10 nodes).

We have tested four different start distributions of load.

In start distribution (1), all the load (10,000 units) is placed on a single, special node—namely, the most connected node (node 0). This node has $k_{max} = 2203$ neighbors. Hence, for tests with plain diffusion we set $c = 1/2500$, and we used the same value for c_3 in the chemotaxis tests.

Start distribution (2) has all load on a single, rather poorly connected node (node 100). Hence we expect convergence time to be greatest for this start distribution.

Start distribution (3) is generated by taking 10,000 units of load, and placing one unit at a time at a randomly selected node, until all 10,000 units have been placed. With this procedure, we typically obtain starting load values ϕ_i^0 in the range between 0 and approximately six units.

Finally, start distribution (4) was used as a test for convergence. To generate this start distribution, all nodes were first given exactly one unit of load. Then, for each node, a random number between -0.01 and $+0.01$ is found, and this amount of load is transferred to another, randomly chosen node. As a result of this procedure, the node loads for start distribution (4) typically fall in the range 0.94–1.06.

All of our tests were performed using the Peersim simulation platform. For details of the implementation, see D09. Here we mention that time in our simulations is measured in units of cycles. In the first phase of each cycle, all nodes compute their own changes in load (due to sending and receiving), and in signal (from emission, sending, and receiving). In the second phase of the same cycle, each node updates its load ϕ_i and signal S_i, using the computed

changes found in the first phase. Updating is thus highly synchronous.

3.2.2 Load balancing criterion

A node's ability to handle load is referred to as its capacity. To simplify our simulation model, we require that all nodes have the same capacity, the capacity of 1.0.

The task of load balancing is to establish and/or maintain a distribution of loads where nodes receive load according to their capacity.

We adopt the simplification that total load equals total capacity, i.e., loads in the network are balanced when convergence with capacities is achieved.

Typically, as the simulation of a protocol for load balancing progresses, and the system under study approaches convergence, the amounts of load that are moved between nodes tend to become smaller and smaller. To reach a state of absolute convergence, i.e., when all nodes have load equal to capacity, therefore typically takes a long time. Consequently, we propose the following relaxed definition of convergence.

Definition. *Convergence of load balance protocol.* At the end of each simulation cycle, let min be the smallest load value in the network and max be the largest. Also, let $threshold$ be a small number. For each simulation cycle where $(max - min) < threshold$ holds, a convergence counter is incremented by one. Also, for each simulation cycle where $(max - min) >= threshold$ holds, the convergence counter is decreased by 10 or set to zero if a negative value would result. A load balance protocol simulation is then said to converge when the convergence counter reaches the value of 100.

3.2.3 Results of the tests

Here we report only results obtained for the case that the load diffuses according to the constrained version of Eq. (22). The reason for this is that the constrained version outperformed the unconstrained version in nearly every respect. In particular, the constrained version always converged, whereas the unconstrained version often failed to converge.

We have conducted a rather comprehensive set of performance measurements of the constrained version of chemotaxis and are able to report on many of the figures of merit (FOM) for load balancing listed in Deliverable D04. Our evaluation work is not complete, however. We have not yet conducted performance measurements assessing the degree to which chemotaxis exhibits "nice properties", i.e., whether favorable FOM results still hold under conditions of environmental change, damage, or growth. We start by presenting test results and then move on to discuss FOMs.

Most of our testing has focused on observing the effects of various signal diffusion speeds on the time to reach convergence. (All tests involving the constrained load diffusion method use the convergence criterion with a threshold of 0.1). Recall that the Version 6 signal diffusion algorithm allows its speed to be altered by choosing different values for the diffusion constant c_{default}. The fastest signal diffusion speed is always obtained by our Version 10 algorithm. Version 6 with $c_{\text{default}} = 1$ gives the second fastest signal speed. Progressively slower signal speeds are then obtained by halving the value of c_{default}. We chose Version 6 with $c_{\text{default}} =$

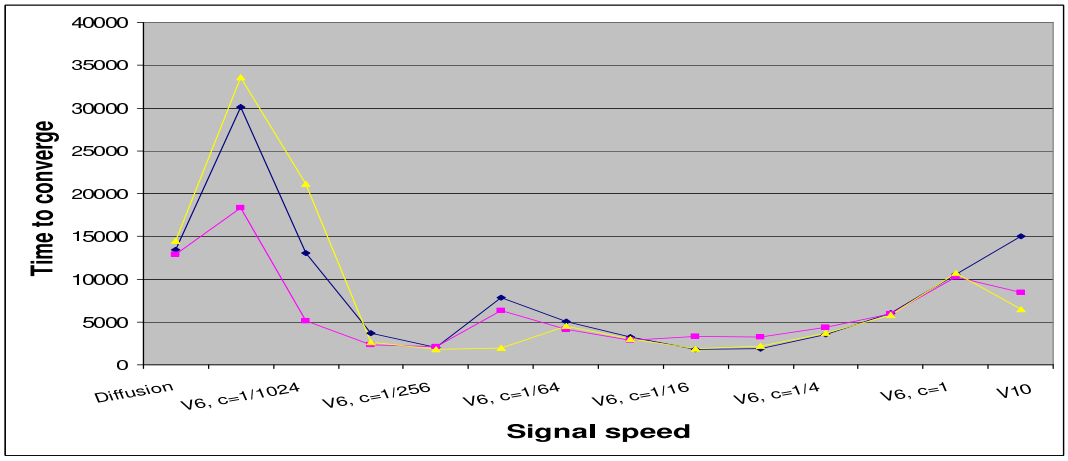


Figure 16: The effect of increasing signal diffusion speeds on time to converge for load that is guided by signal according to the constrained algorithm. Each of the three graphs corresponds to a different instance of start distribution (3). The times to converge for plain diffusion are plotted to the far left. All other plots show convergence times for chemotaxis-based load balancing.

$1/2048$ as our slowest signal diffusion algorithm.

The time to convergence for different signal speeds when using start distribution (3), the random distribution, are plotted in Figure 16. Each of the three graphs represents a different instance of the start distribution. Signal speed increases along the horizontal axis. For ease of comparison, we have plotted on the far left the time to converge for plain diffusion, using the same instances of start distribution (3).

The stability of the constrained version of chemotaxis is evident from the graphs in Figure 16: all tests successfully converged to a balanced load. Chemotaxis-based load balancing also exhibited shorter time to convergence than did plain diffusion for most tests. Several signal speeds produced reductions in convergence times of about 80%[3]. Chemotaxis performed worse than plain diffusion when signal speeds were very slow. Finally, we note that the use of version 10 for signal is clearly not optimal. For one thing, the convergence times are higher than for most of the version 6 results. Also, when signal diffused using the Version 10 algorithm, an increased sensitivity to variations in start distribution was observed, as compared to Version 6 algorithms at the speedier end of the spectrum.

In order to gain a deeper understanding of the behavior of chemotaxis-based load balancing algorithms, we have inspected the plots of maximum and minimum load values in the system

[3]The shortest time to convergence for chemotaxis was 1795 cycles. Note that the fastest convergence time observed for single-component, plain diffusion (using the Version 6 algorithm with $c_{\text{default}} = 1$) was 1643 cycles.

as a function of time. We found huge variations between how different signal diffusion algorithms impact the progression towards a balanced load. Some algorithms exhibit an extremely rapid progression towards convergence during the first few hundred cycles, only to require thousands of cycles to satisfy our definition of a balanced system. Other algorithms result in a steady progression towards convergence. Also, the observed degree of oscillations in maxima over time is highly variable across algorithms. We therefore suspect that the convergence times reported in Figure 16 are highly dependent on our particular definition of convergence. For example, had we defined convergence as when the difference between the maximum and minimum load values is less than 2.0, say, then by far the shortest time to convergence would have been obtained by using the Version 6 algorithm with $c_{\mathrm{default}} = 1/2048$ as the signal diffusion algorithm.

Figure 17 shows total load moved to reach convergence for each of the tests plotted in Figure 16. From this figure we see that, even though chemotaxis-based load balancing offered significant improvements in convergence times in the above tests, those same tests did not show significant reductions in total load movement required to reach convergence compared to plain diffusion[4]. We observed no relationship between convergence times and total load moved, but did observe a clear indication of an increase in total load moved with increasing signal speed. Finally, we note that in this figure, as in Figure 16, the version 10 chemotaxis results show both a poor value for the FOM (here total load moved), and a relatively high sensitivity to small variations in start distribution.

Chemotaxis-based load balancing addresses systems where the ability to move load is limited. Therefore, we may argue that what is more important than the total load moved is the amount of load that is moved during a short interval, e.g., during a single cycle. Figure 18 shows the largest amount of load that was moved in a single cycle during each experiment. At the time of writing, data about load movements have only been collected for two instances of start distribution (3), giving two graphs instead of three. We note that the largest load amount moved for plain diffusion is missing from Figure 18; for both start distribution instances, this amount is 0.002. Thus we see that version-6 chemotaxis and plain diffusion have roughly equal values for this FOM, while (again) version-10 chemotaxis is much worse, and more sensitive.

From Figure 18 we observe that loads guided by the speedier instances of the Version 6 signal diffusion algorithm produced smaller values for the largest single-step load amount moved. The smallest maximum is 0.014[5]. It is interesting to note the relatively poor performance resulting from using the Version 10 algorithm to diffuse signal: the maximum load amount moved in a single cycle is about 50 times higher than for the best cases. Relative differences among different instances of the Version 6 algorithm are also rather big (up to a factor of 7).

Next we summarize the results obtained to date in terms of the FOMs described in D04. We report here work which is not finished; but we have results to report for many of the FOMs. What we lack is complete studies for these FOMs, as well as an evaluation of the nice properties.

[4]Note that single-component, plain diffusion using the Version 6 algorithm with $c_{\mathrm{default}} = 1$ (which exhibits convergence times similar to the shortest times obtained by chemotaxis-based algorithms) in total moves about 27.000 load units, i.e., significantly more than all but one of our chemotaxis experiments.

[5]Note that single-component, plain diffusion using the Version 6 algorithm with $c_{\mathrm{default}} = 1$ (exhibiting convergence times similar to the shortest times obtained by chemotaxis-based algorithms) moves up to 5.0 units of load in a single cycle. (The largest load value in the random distribution instance in question is 6.0, giving a maximum difference between a node's load and capacity of 5.0.)

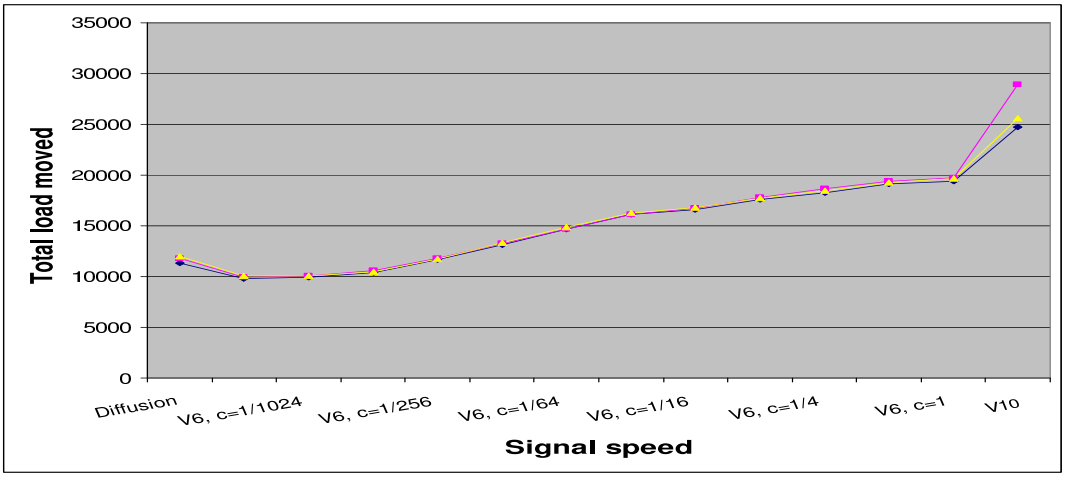


Figure 17: Total load moved to reach convergence as a function of signal speed. Each of the three graphs corresponds to a different instance of start distribution (3). Numbers for plain diffusion are plotted on the far left.

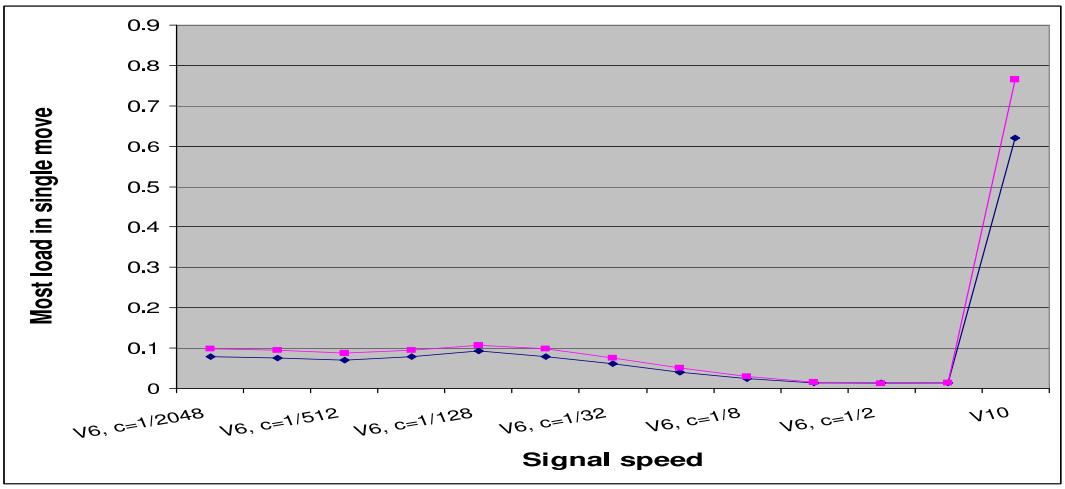


Figure 18: Largest load amount moved during any single cycle, over an entire simulation run, as a function of signal speed. Each graph represents a different instance of start distribution (3).

3.2.4 Convergence

Here our conclusions are rather clear: the constrained version of chemotaxis converges reliably, when signal diffuses according to version 6 (for all values of c_{default}), or according to version 10. The unconstrained version, in contrast, does not converge reliably; hence we do not consider it further.

3.2.5 Approach to convergence

We have clear impressions of rate of convergence, but do not yet have systematic quantitative measures. Hence we report qualitative comments here, taken from a large number of tests.

We note first that version-10 chemotaxis has a relatively slow approach to convergence over the entire time interval.

Version-6 chemotaxis, in contrast, displays a rather interesting behavior. That is, the *slowest* forms of version-6 chemotaxis (ie, those with the smallest values for c_{default}) show the *fastest* approach to convergence. Their convergence time, as shown in Figure 16, is high, but this is only because of our relatively strict convergence criterion. If we regarded all loads falling within $\pm$100% of capacity as converged, for example, then the version-6 results with very small c_{default} would have by far the lowest convergence time.

Thus we find that, for this FOM, version 10 is again uninteresting, while version 6 is both promising and puzzling.

3.2.6 Convergence time

Our results for the random initial distribution (3) are shown in Figure 16. Our general conclusion here is rather clear: version-6 chemotaxis is best, at least for most values of c_{default}. In particular, the best results for version 6 are many times (about 6) smaller than those for unaided diffusion. Version 10 appears not to be competitive with version 6, although it in two of three cases showed an improvement over unaided diffusion.

These conclusions hold also for our tests using the initial distribution (1), with all load placed on a poorly connected node. Further tests are needed to verify that these conclusions are fully general.

3.2.7 Total load moved

Here we refer to Figure 17. We see here that, for almost all cases, chemotaxis moves more load than does plain diffusion. Ignoring version 10, we see that the total load moved for chemotaxis ranges from about 80% to about 160% of that amount moved using plain diffusion.

For the less realistic case where all load is placed initially on one node, we find the opposite to be true: plain diffusion moves about 370,000 units of load to convergence, while the values for version 6 range from about 70,000 to about 125,000, as c_{default} is varied. (Version 10 is not competitive here either.)

Hence we find a mixed picture for the ranking of chemotaxis vs. plain diffusion for this FOM.

3.2.8 Maximum load moved in one step

Here we refer to Figure 18. Version 10 is very poor, while version 6 and plain diffusion give roughly the same results. This suggests that (at least, for this topology) the increase in convergence rate obtained from chemotaxis may also be realized for systems with a strict upper bound on the load sent.

Our results for initial distribution (1) are different from those in Figure 18. Here plain diffusion moves about 4 units in one time step, while the version-6 values vary smoothly between about 90 and about 5. That is, the version-6 results for this initial distribution are rather sensitive to the parameter c_{default}, and only the best results are comparable with those for plain diffusion.

3.2.9 Sensitivity to initial distribution

This FOM is a kind of nice property. It is clear from all the above that both plain diffusion and version-6 chemotaxis are in general rather insensitive to variations in start distribution (3), while version-10 chemotaxis is clearly more sensitive. This statement holds for essentially all of the FOMs we have looked at. This result is of course very tentative.

3.3 Summary and outlook

It is clear from the above that our chemotaxis project has at least reached the proof-of-principle stage. We believe furthermore that the principle has been well demonstrated: our results show that use of signaling can significantly enhance the speed of convergence of slow-moving load, with little or no penalty in total load moved. In particular, the results of version-6 chemotaxis are promising, both for the FOMs we have seen, and for the insensitivity of these FOMS for this algorithm to small variations in initial distribution.

Clearly, the program of evaluation described in D04 remains to be completed for the chemotaxis method. The tests reported here are not thorough enough. The principal outstanding tasks remaining to be done are the evaluation of these approaches on other topologies (pseudo-random, grid), and the evaluation of nice properties. These tasks are a high priority for the remaining year of BISON.

We believe that plain diffusion is a good reference system, in that it is simple, requires no global knowledge, and—as long as one avoids the unstable range of high diffusion speed—always converges to the uniform state. Hence we would like to compare as many approaches as possible to this reference system. Doing so requires finding a fairness criterion or criteria for each such approach—something which we expect to be possible in most cases. In particular, if time permits, we will also develop a fairness criterion which allows the comparison of the two approaches reported here with the pairwise load balancing approach described in Section 3.2 of D08.

4 T-Man: Construction of Large-Scale Overlay Topologies

Overlay topology plays an important role in many P2P systems. Topology serves as a basis for achieving functions such as routing, searching and information dissemination, and it has a major impact on their efficiency, cost and robustness. Furthermore, the solution to problems such as sorting and clustering of nodes can also be interpreted as a topology. In this light, we consider the construction and maintenance of overlay topologies an advanced service, that potentially relies on other services and more basic topologies (such as the random topology) and that can directly serve important applications.

We define the *topology construction problem* in overlay networks and present a simple protocol called T-MAN to solve it. T-MAN represents a unified gossip-based framework in which a large class of topologies can be evolved starting from a random network simply by specifying an appropriate *ranking function*. The protocol can be applied as a standalone solution as well as a component for recovery or bootstrapping of other protocols.

Note that structured overlay topology management was not part of the original list of advanced services, so this service was not part of deliverable D08. However, due to its (originally underestimated) importance, we have devoted significant efforts to this field and plan to continue to do so.

4.1 Introduction

The advanced services studied in the BISON project include distributed processing, storage and content sharing. A crucial component of these services in P2P systems is an overlay network, that makes it possible for the participants of a fully distributed system to communicate and organize information. For example, in the case of content sharing, there are multiple ways of deploying overlay networks to facilitate search and information flow. These include superpeer topologies, semantic clustering, and distributed hashtables (DHTs).

These topologies have to be maintained and constructed in large scale highly dynamic scenarios. This task is a central and challenging problem, that was underestimated in the original technical plan of the project. During the second year, we have started to pay more attention to the problem of overlay networks.

We consider topology management as a general purpose function that is desirable in large distributed systems. We define the *topology construction problem* as a cornerstone of topology management, and propose a gossip-based probabilistic solution to this problem called T-MAN. The desired topology is described using a single ranking function that all nodes can apply to order any subset of potential neighbors according to preference for actually being selected as a neighbor. Using only local gossip messages, T-MAN gradually evolves the current topology towards the desired target structure with the help of the ranking function.

The key idea of the protocol originates from a biological analogy: the process of *adhesion*. Due to adhesion, different types of cells interact with each other: some types prefer to be close, some types avoid each other. As a result, patterns emerge that organize different cell types into shapes defined by the exact parameters of the adhesive components [29, 11].

We show experimentally that the protocol is scalable and fast, with convergence times that

grow only as the logarithm of the network size. These properties allow T-MAN to be practical even when several different topologies have to be created on demand, and also in dynamic systems where the set of nodes or their properties change rapidly. Additionally, the general formulation of the ranking function allows us to deal with a wide range of different topologies.

Although this work is concerned mainly with exploring the basic properties of T-MAN by examining simple topologies like ring, mesh and binary tree, the protocol does have a wide range of applications in implementing advanced services. We briefly outline three such applications: sorting, clustering and a distributed hash table (DHT) based on an approximate Pastry topology [28].

Related work Gossip-based protocols have gained notable popularity in various contexts [31, 3, 5]. We suggest a novel application of the gossip communication model to solve the topology construction problem. Issues related to topology management itself have also received considerable attention. Examples from the vast literature include DHTs [28, 35, 30], unstructured overlays [26, 13], and superpeer topologies [33]. As for topology construction, Massoulié and Kermarrec [20] propose a protocol to evolve a topology that reflects proximity, Voulgaris and van Steen [32] propose a method to jump-start Pastry. Unlike these specific solutions, T-MAN is a generic framework and can be used to construct a large class of different topologies quickly in a simple and scalable manner.

4.2 The problem

We assume that we are given a (perhaps random) overlay network, and we are interested in constructing some desirable topology by connecting all nodes in the network to the right neighbors. The topology can be defined in many different ways and it will typically depend on some properties of the nodes like geographical location, semantic description of stored content, storage capacity, etc. We need a formal framework that is simple yet powerful enough to be able to capture most of the interesting structures. Our proposal is the *ranking function* that defines the target topology through allowing all nodes to sort any subset of nodes (potential neighbors) according to preference to be selected as their neighbor.

For a more formal definition, let us first define some basic concepts. We consider a set of nodes connected through a routed network. Each node has an address that is necessary and sufficient for sending it a message. Nodes maintain addresses of other nodes through *partial views* (*views* for short), which are sets of c *node descriptors*. In addition to an address, a node descriptor contains a *profile*, which contains those properties of the nodes that are relevant for defining the topology, such as ID, geographical location, etc. The addresses contained in views at nodes define the links of the *overlay network topology*, or simply the *topology*. Note that parameter c defines the node degree of the overlay network and is uniform for all nodes.

We can now define the *topology construction problem*. The input of the problem is a set of N nodes, the view size c and a *ranking function* R that can order a list of nodes according to preference from a given node. The ranking function R takes as parameters a base node x and a set of nodes $\{y_1, \ldots, y_m\}$ and outputs a set of orderings of these m nodes. The task is to construct the views of the nodes such that the view of node x, denoted view$_x$, contains exactly the first c elements of a "good" ranking of the entire node set, that is, $R(x, \{\text{all nodes except } x\})$

contains a ranking that starts with the elements of view$_x$.

One (but not the only) way of obtaining ranking functions is through a distance function that defines a metric space over the set of nodes. The ranking function can simply order the given set according to increasing distance from the base node. Let us define some example distance-based topologies of different characteristics. From now on, to simplify our language and notation, we use the nodes and their profiles interchangeably. In the case of the line and ring topologies, the profile of a node is a real number. The distance function for the line is $d(a, b) = |a - b|$. In the case of a ring, profiles are from an interval $[0, N]$ and distance is defined by $d(a, b) = \min(N - |a - b|, |a - b|)$ Ranking is defined through this distance function as described above. This construction can be easily generalized to arbitrary dimensions to get for example a mesh or a torus. A low diameter topology can be constructed from a binary tree: the profiles are binary strings of length m, excluding the all zero string. Distance is defined as the shortest path length between the two nodes in the following undirected rooted binary tree. The string $0 \dots 01$ is the root. Any string $0a_2 \dots a_m$ has two children $a_2 \dots a_m 0$ and $a_2 \dots a_m 1$. Strings starting with 1 are leafs. This topology is of interest because (unlike the previous ones) it has a very short (logarithmic) diameter of $2m$.

4.3 The proposed solution

The topology construction problem becomes interesting when c is small and the number of nodes is very large. Randomized, gossip-based approaches in similar settings, but for other problem domains like information dissemination or data aggregation, have proven to be successful [5, 14]. Our solution to topology construction is also based on a gossip communication scheme.

We assume that nodes have access to local clocks that measure the passage of real time. We make the simplifying assumption that there exists a single synchronization point when the protocol is started at all nodes. Furthermore, we assume that the communication channels and the nodes are reliable. These assumptions allow us to focus on the counter-intuitive, and therefore, more interesting properties of our protocol. We don not believe that these assumptions are too restrictive: in similar gossip-based applications we have proposed a solution to relax them [22].

Each node executes the same protocol shown in Figure 19. The protocol consists of two threads: an active thread initiating communication with other nodes, and a passive thread waiting for incoming messages.

A view is a set of node descriptors. A call to MERGE(view$_1$,view$_2$) returns the union of view$_1$ and view$_2$. Method INITIALVIEW() is used to set the initial view of the node and is part of bootstrapping. It is desirable that this initial view be as close as possible to a random sample of c nodes drawn from the entire node set. Fortunately there are a number of protocols that provide us with such a service [13].

The two key methods are SELECTPEER and SELECTVIEW. Method SELECTPEER uses the current view to return an address. First, it applies the ranking function to order the elements in the view. Next, it returns a random sample from the *first half* of the view according to ranking order. This choice of implementation is motivated in [12] where we analyze the convergence of the protocol. Method SELECTVIEW(BUFFER) also applies the ranking function to order the

```
view ← initialView()
do at a random time once in each
consecutive interval of T time units
  p ← selectPeer()
  myDescriptor ← (myAddress,myProfile)
  buffer ← merge(view,{myDescriptor})
  send buffer to p
  receive view_p from p
  buffer ← merge(view_p,view)
  view ← selectView(buffer)
```

(a) active thread

```
do forever
  (q, view_q) ← waitMessage()
  myDescriptor ← (myAddress,myprofile)
  buffer ← merge(view,{myDescriptor})
  send buffer to q
  buffer ← merge(view_q,view)
  view ← selectView(buffer)
```

(b) passive thread

Figure 19: The T-MAN protocol.

elements in the buffer. Subsequently, it returns the *first c elements* of the buffer according to ranking order.

The underlying idea is that in this manner nodes improve their views using the views of their current neighbors, so that their new neighbors will be "closer" according to the target topology. Since all nodes do the same concurrently, neighbors in the subsequent topologies will be gradually closer and closer. This also means that the views of the neighbors will keep serving as a useful source of additional, even better links for the next iteration. Still, the behavior of the protocol is not easy to grasp since this highly interdependent dynamics of the views is rather non-trivial. In [12] we offer a fairly simple and intuitive way of modeling and understanding the protocol.

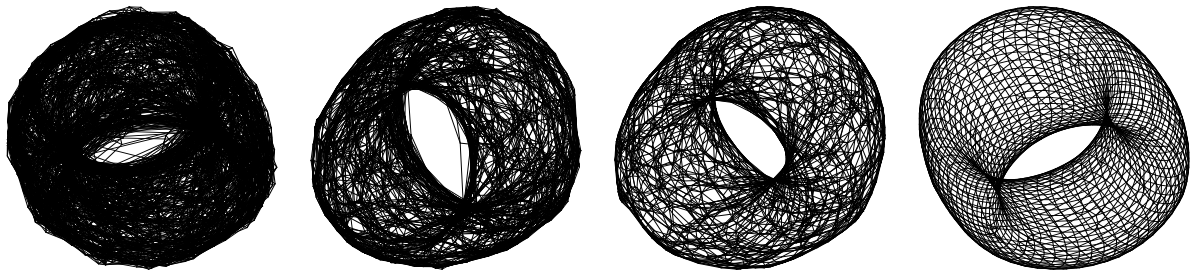
Although the protocol is not synchronous, it is often convenient to refer to *cycles* of the protocol. We define a cycle to be a time interval of $T/2$ time units where T is the parameter of the protocol in Figure 19. Note that during a cycle, each node is updated once on the average.

Figure 20 illustrates the results of the protocol when used to construct a small torus (visualizations were done using [17]). For this example, it is clear that 15 cycles are sufficient for convergence, and the target topology is already evident even after very few cycles. As we will see, T-MAN proves to be extremely scalable and the time complexity of the protocol remains in this order of magnitude even for a million nodes.

4.4 Simulation Experiments

All the simulation results presented here were produced using PEERSIM, an open-source simulator developed at the University of Bologna [27].

We examine the three distance-based ranking functions that define the ring, torus and binary tree topologies, as defined in Section 4.2. The motivation of this choice is that the ring is a large diameter topology and it is relevant for the sorting application (Section 4.5.1), the binary tree is of a logarithmic diameter and the torus is relevant in proximity problems being based on



after 3 cycles after 5 cycles after 8 cycles after 15 cycles

Figure 20: Illustrative example of constructing a torus over $50 \times 50 = 2500$ nodes, starting from a uniform random topology with $c = 20$. For clarity, only the nearest 4 neighbors (out of 20) of each node are displayed.

a 2-dimensional grid. The network sizes (N) examined are 2^{14}, 2^{17} and 2^{20}. We initialize the profiles of the nodes in a regular manner, that is, in the case of the ring topology, we assign the numbers $1, 2, \ldots, N$ to the nodes, and likewise for the torus $((1,1),(1,2),\ldots,(\sqrt{N},\sqrt{N}))$ and the binary tree (all binary strings of length $\log_2 N$).

This regularity is not critical for the success of the protocol. On the contrary, one of the important applications is sorting an arbitrary set of numbers, as we argue in Section 4.5.1. However, this controlled setting allows us to monitor the dynamics of the protocol in a more informed manner as the distance function becomes equivalent to the hop count in the *target topology* defined by the links that connect nodes at distance 1 (the target links). During the experiments we focus on the dynamics of the number of target links that are found. As a measure of performance, the *convergence factor* is defined as the factor by which the number of target links found increases from one cycle to the next. Note that a constant convergence factor means exponential increase.

Views were initialized using the NEWSCAST protocol, an implementation of the peer sampling service based on a gossip-based unstructured overlay [13]. This is to ensure that the convergence results are not due to the uniform random sampling in the initial view (which is a rather strong assumption), but can also be obtained using realistic starting conditions.

The results are shown in Figure 21. To interpret the results, we refer to our technical report [12] where we suggest an approximate model to describe the speed of convergence. The model predicts that the number of cycles i needed to include all target links in the view of all nodes is given as $i = \log_2(N-1) - \log_2 c$, where c is the view size and N is the network size. The model also predicts that until this point, the convergence factor is constant and equals 2. The table below illustrates the predicted number of cycles to reach the perfect solution.

	$c = 20$	$c = 40$	$c = 80$
$N = 2^{14}$	10	9	8
$N = 2^{17}$	13	12	11
$N = 2^{20}$	16	15	14

In Figure 21 we can clearly observe this trend of a one-cycle shift when doubling the view size and a three-cycle shift when increasing the network size eight fold. The only exception is for

$c = 20$ which is shifted slightly more than predicted. This is due to the fact that $c = 20$ is small enough so that sampling errors can play a more important role in undermining the validity of the prediction. On the other hand, along the dimension of network size with a fixed $c = 20$, the scaling is still according the the three cycle (and so logarithmic) shift as predicted. Remarkably, the above observations hold for all three of the topologies.

When the topology has already converged, the few nodes that are still incorrectly linked can be though of as "climbing" on the converged structure during the consecutive cycles of T-MAN. This means that in this end phase convergence time does depend on the target topology. Accordingly, in the binary tree topology, we observe rapid convergence. In fact, logarithmic convergence, because the evolved structure allows for efficient routing, being low diameter. Similar arguments hold for the torus, only the convergence time there is not logarithmic but grows with the square root of the network size in the worst case. In both cases, we can observe fast convergence even for the smallest view size. The case of the ring is different, because the target topology has a large diameter that grows linearly with the network size, so the remaining few misplaced nodes can have a hard time reaching their destination. In the largest network, even with $c = 80$ we have not reached full convergence in the single run we performed within 100 cycles. Note however that at cycle 30 the number of missing links is already less then 10, out of the $2 \cdot 2^{20} = 2,097,152$ target links, and even for $c = 40$ we miss only 100 links in cycle 100. This level of precision will satisfy most applications.

4.5 Application Examples

The primary goal of this section is to underline the generality of the approach by outlining the main ideas in using T-MAN to solve some potentially important applications.

4.5.1 Clustering and Sorting

So far we have considered rather artificial settings to understand the behavior of T-MAN better. In particular, the profiles of the nodes were initialized in a regular manner. In practice this will hardly happen. Typically, the properties of the nodes follow some specific distribution. This distribution can also be "patchy", there can be dense clusters separated by unrepresented regions in the profile space. Very informally, when applying T-MAN in such a setting using a simple distance-based ranking function, the resulting topology will be clustered and most likely disconnected, because nodes in the same cluster will eventually be linked to each other only (see Figure 22). In many applications, like clustering based on semantic, geographic or any other definition of proximity, this property can be exploited to find the desired clusters.

In the case of the sorting problem, where we would like to connect each node to those nodes that directly preceed and follow them according to some total ordering relation, we need to prevent clustering. This can be achieved by the following direction dependent ranking. First, separate the set of nodes to be ranked into two groups: one that is to the left, and another that is to the right of the base node. Order these two sets according to the underlying desired ordering. Merge the ordered sets so that a node that had index i in any of the sets is assigned index $2i$ or $2i+1$ in the final ranking, choosing randomly between these two possibilities. Applying this ranking function makes it possible to practically reduce the sorting problem to the one dimen-

sional topology construction problem that we have studied extensively in Section 4.4. The effect of direction dependent ranking is illustrated in the small example in Figure 22. Observe the clustering with the distance-based ranking and the perfect ordering with direction dependent ranking. Direction dependent ranking can be easily extended to other problems, for example, creating a *connected* topology in two dimensions that reflects geographical proximity.

4.5.2 A DHT

As yet another application, we present a simple way of evolving a DHT topology with T-MAN. The target topology is inspired by Pastry [28]. The ranking function that generates this topology is based on a distance function. The trick in defining this distance function is similar to the one we have seen in the case of binary trees: we define an ideal undirected graph over the set of ID-s that are m-bit binary numbers (bit strings of length m) and we define the distance of two ID-s as the length of the shortest path in this graph. We start with a directed graph by linking bit string $x_1 \dots x_m$ to the following m strings: $\overline{x_1}x_2 \dots x_m$, $x_1\overline{x_2}x_3 \dots x_m$, $x_1x_2\overline{x_3}x_4 \dots x_m$, etc., where $\overline{x_i} = 1 - x_i$. To get the undirected graph we simply drop the directionality of the edges.

As a backup, we also evolve a sorted ring topology using another instance of T-MAN as described above, to maximize the probability that routing is successful. The routing table is composed of the neighbors in these two topologies, and the next hop for a target is selected based on numeric difference between the ID of the target and the table entries (now interpreting ID-s as numbers). We require strictly decreasing difference to avoid loops. The links from the ring topology are used only if no suitable links are available in the Pastry-inspired topology.

Figure 23 illustrates the convergence of the routing performance while the topology is being evolved, starting from random routing tables. We can observe that the number of missed targets quickly becomes insignificant (from cycle 23 only 3 cases out of the 5300 shown in the figure), and the hop count of both the successful and unsuccessful routes remains very low.

4.6 Summary and outlook

We have presented a protocol for topology construction, T-MAN, that is simple, general and fast. Simplicity makes it easier to implement, debug and understand. Generality allows it to be applied as an off-the-shelf component for prototyping or even as a production solution that could be implemented even before the final desired topology is known. In fact, the ranking function can be generated dynamically by users, or by some monitoring service, and the corresponding topology can be created on the fly. Finally, speed makes it possible to construct a topology quickly from scratch (recovery from massive failures or bootstrapping other protocols on demand) or where topology maintenance is in fact equivalent to the continuous re-creation of the topology (for example, due to massive churn).

Our current work is towards the application of T-MAN for jump-starting existing DHT implementations and providing them robustness in the presence of massive failures and extreme churn. We are also continuing our study of T-MAN at an abstract level to better understand its behavior and characterize its scope and performance.

5 Conclusion

In this deliverable we have reported detailed evaluation of the different advanced services we have developed for $p2p$ networks. We believe that any service built on dynamical networks like $p2p$ network should be evaluated on basis of two broad criteria - (1) the traditional figures of merit and (2) 'nice' properties like robustness, scalability etc. All these evaluation tests have been mostly performed for immune-based search algorithm. Moreover, important theoretical advances have been made to understand the reasons behind its better performance.

The work on chemotaxis-based load balancing is also moving in that direction. The proof-of-principle is well demonstrated. The extensive evaluation work will be completed in the last year. The concept of evolving DHT with T-MAN is also showing promising results.

Keeping all these developments in mind, we believe, the last year of BISON will produce exciting new results and open up new possibilities in the domain of advanced services.

References

[1] L A Adamic, R M. Lukose, A R. Puniyani, and B A Huberman. Search in power-law networks. *Physical Review E*, 64:046–135, 2001.

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

[3] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database management. In *Proc. of PODC'87*, pages 1–12, Vancouver, August 1987. ACM.

[4] A Deutsch, N Ganguly, G Canright, M Jelasity, and K Monsen. Models for advanced services in AHN, P2P Networks. Bison Deliverable, www.cs.unibo.it/bison/deliverables/D08.pdf, 2003.

[5] Patrick Th. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulié. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, May 2004.

[6] N Ganguly, L Brusch, and A Deutsch. Design and analysis of a bio-inspired search algorithm for peer to peer networks. In *Post Proceeding, Self-Star Conference (submitted)*, June 2004.

[7] N Ganguly, G Canright, and A Deutsch. Design of a robust search algorithm for p2p networks. In 11^{th} *International Conference on High Performance Computing*, December 2004.

[8] N Ganguly, G Canright, and A Deutsch. Design of an efficient search algorithm for p2p networks using concepts from natural immune systems. In 8^{th} *International Conference on Parallel Problem Solving from Nature*, September 2004.

[9] N Ganguly and A Deutsch. A cellular automata model for immune based search algorithm. In 6^{th} *International conference on Cellular Automata for Research and Industry*, October 2004.

[10] N Ganguly and A Deutsch. Developing efficient search algorithms for p2p networks using proliferation and mutation. In 3^{rd} *International Conference on Artificial Immune Systems*, September 2004.

[11] J. A. Glazier and F. Graner. Simulation of the differential adhesion driven rearrangement of biological cells. *Phys. Rev. E*, 47(3):2128–2154, 1993.

[12] Márk Jelasity and Ozalp Babaoglu. T-Man: Fast gossip-based construction of large-scale overlay topologies. Technical Report UBLCS-2004-7, Univ. of Bologna, Italy, May 2004. `http://www.cs.unibo.it/techreports/2004/2004-07.pdf`.

[13] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In Hans-Arno Jacobsen, editor, *Middleware 2004*, volume 3231 of *LNCS*. Springer, 2004.

[14] Márk Jelasity and Alberto Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proc. of ICDCS 2004*, pages 102–109, Tokyo, Japan, 2004.

[15] C Jin, Q Chen, and S Jamin. Inet: Internet topology generator. University of Michigan Technical Report CSE-TR-433-00, 2002.

[16] M. A. Jovanovic, F. S. Annexstein, and K. A. Berman. Scalability issues in large peer-to-peer networks - a case study of gnutella. Technical Report University of Cincinnati, 2001.

[17] Yehuda Koren. Embedder. `http://www.research.att.com/~yehuda/index_programs.html`.

[18] Dik L. Lee, Huei Chuang, and Kent Seamons. Document ranking and the vector-space model. *IEEE Softw.*, 14(2):67–75, 1997.

[19] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 258–259. ACM Press, 2002.

[20] Laurent Massoulié, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Network awareness and failure resilience in self-organising overlays networks. In *Proc. of SRDS 2003*, pages 47–55, Florence, Italy, 2003.

[21] A Medina, A Lakhina, I Matta, and J Byers. Brite: An approach to universal topology generation. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS*, August 2001.

[22] Alberto Montresor, Márk Jelasity, and Ozalp Babaoglu. Robust aggregation protocols for large-scale overlay networks. In *Proc. of DSN 2004*, pages 19–28, Florence, Italy, 2004. IEEE Computer Society.

[23] J. D. Murray. *Mathematical Biology*. Springer-Verlag, 1990.

[24] A. Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O Reilly Books, 2001.

[25] G. Pandurangan, P. Raghavan, and E. Upfal. Building low-diameter peer-to-peer networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 21(6), 2003.

[26] Gopal Pandurangan, Prabhakar Raghavan, and Eli Upfal. Building low-diameter peer-to-peer networks. *IEEE JSAC*, 21(6):995–1002, August 2003.

[27] PeerSim. http://peersim.sourceforge.net/.

[28] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In Rachid Guerraoui, editor, *Middleware 2001*, volume 2218 of *LNCS*, pages 329–350. Springer, 2001.

[29] M. S. Steinberg. Does differential adhesiveness govern self-assembly processes in histogenesis? Equilibrium configurations and the emergence of a hierarchy among populations of embryonic cells. *J. Exp. Zool.*, 173:395, 1970.

[30] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM 2001*, pages 149–160, San Diego, CA, 2001. ACM, ACM Press.

[31] Robbert Van Renesse, Kenneth P. Birman, and Werner Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM TOCS*, 21(2), May 2003.

[32] Spyros Voulgaris and Maarten van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In *Proc. of DSOM 2003*, number 2867 in LNCS. Springer, 2003.

[33] Beverly Yang and Hector Garcia-Molina. Designing a super-peer network. In *Proc. of ICDE 2003*, Los Alamitos, CA, March 2003. IEEE Computer Society.

[34] S. B. Yuste and L. Acedo. Number of distinct sites visited by n random walkers on a euclidean lattice. *Physical Review E*, 61:6327–34, 2000.

[35] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, Univ. of California, Berkeley, April 2001.

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

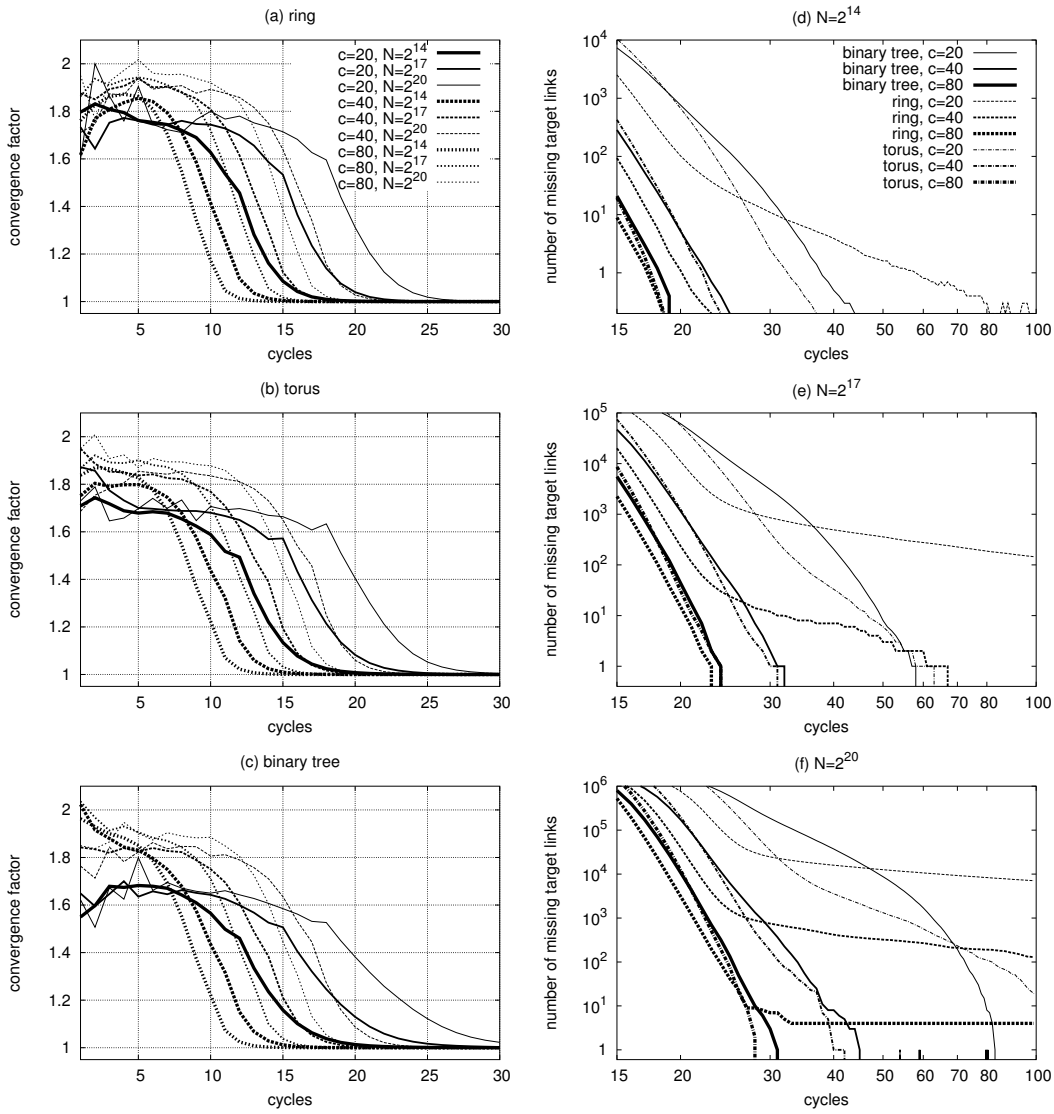
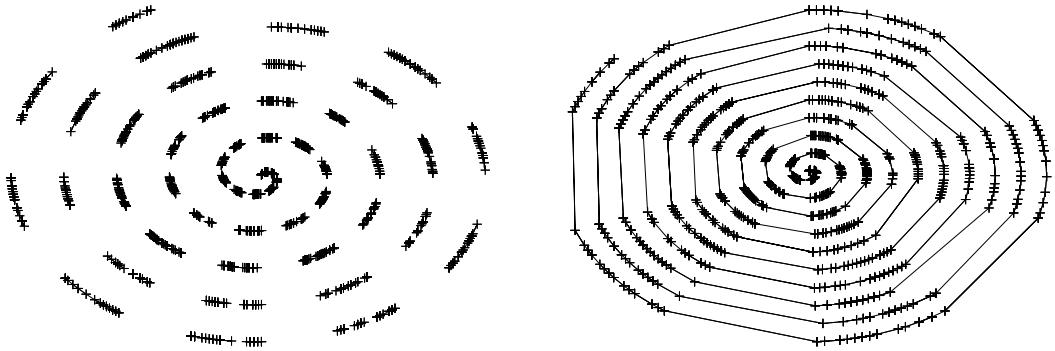


Figure 21: Comparison of convergence speed in the initial exponential phase and in the end phase for network sizes $N = 2^{14}, 2^{17}, 2^{20}$ and $c = 20, 40, 80$ for the ring, torus and binary tree topologies. The results displayed are averages of 50 and 10 runs for $N = 2^{14}$ and $N = 2^{17}$, respectively, and show a single run for the case $N = 2^{20}$.



(a) distance based (b) direction dependent

Figure 22: Illustrative example of converged topologies obtained with distance-based and direction dependent ranking, with $N = 1000$, $c = 20$. The line is displayed as spiral for convenience. Only the closest 2 links are shown from each node.

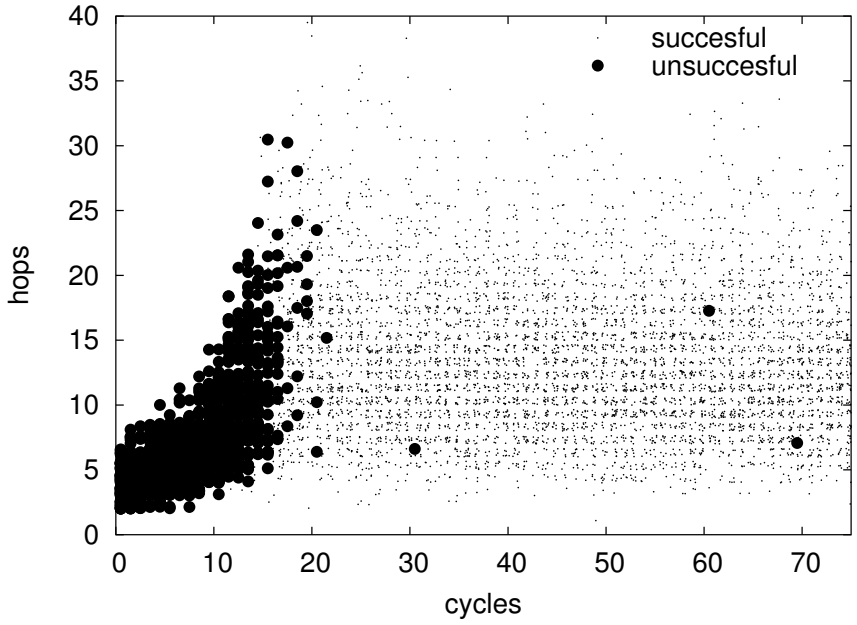


Figure 23: Hop count and success of routing as a function of time (cycles). Each point represents the result of the routing algorithm from a random node to another random node. In each cycle 100 experiments are shown, with a small random translation. Node ID-s are random 62 bit integers, network size is $2^{20} (> 10^6)$, size of routing table is 60: 30 from the Pastry-inspired topology, 30 from the ring.