Project Number 001907

DELIS

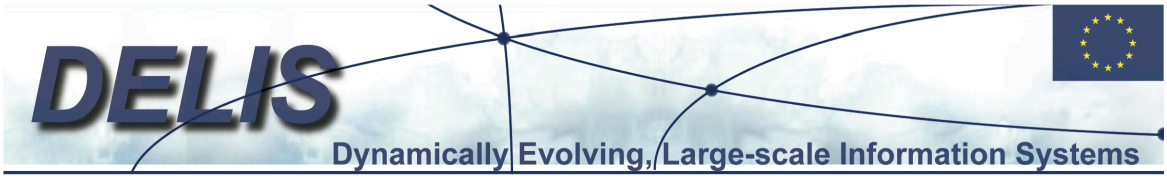
Dynamically Evolving, Large-scale Information Systems

Integrated Project

Member of the FET Proactive Initiative **Complex Systems**

Deliverable D5.3.1

From biological and social algorithms to engineering solutions



Start date of the project:	January 2004
Duration:	48 months
Project Coordinator:	Prof. Dr. math. Friedhelm Meyer auf der Heide Heinz Nixdorf Institute, University of Paderborn, Germany
Due date of deliverable:	December 2006
Actual submission date:	December 2006
Dissemination level:	PU – public
Work Package 5.3:	Biologically and Socially Inspired Design for Dynamic Solution Spaces
Participants:	Universita di Bologna (UniBO), Italy RadioLabs (RAL), Universita di Roma "Tor Vergata", Italy Universitat Pompeu Fabra (UPF), Barcelona, Spain Telenor Communication AS (Telenor), Oslo, Norway
Authors of deliverable:	David Hales (hales@cs.unibo.it) Ozalp Babaoglu (babaoglu@cs.unibo.it) Stefano Arteconi (arteconi@cs.unibo.it) Andrea Marcozzi (marcozzi@cs.unibo.it) Giovanni Cortese (g.cortese@computer.org)

Abstract

This report comprises the complete D5.3.1 deliverable as specified for workpackage WP5.3 in Subproject SP5 of the DELIS (Dynamically Evolving Large-scale Information Systems) Integrated Project.

The essential goal of the DELIS project is to understand, predict, engineer and control large evolving information systems. The main aim of this workpackage is to understand how algorithms originating within biological and social sciences can be applied to applications in distributed computer systems. It is not our aim to present finished applications but to attempt to gain some generic insights that might inform a method by which other such algorithms can be modified. In section 1 we outline our current ideas concerning generic / methodologic issues

However, in order to learn such lessons it is necessary to select specific examples and work these through from abstract model to application model. We selected two broad application domains: broadcasting (section 2) and content replication (section 3). We specified scenarios which captured significant aspects of the domains yet still represent abstractions over a range of possible specific implementations. This is an intermediate step between abstract models and specific implementation level models[1].

[1] Most papers produced within DELIS are available from the DELIS website as DELIS Technical Reports. Where this is the case references are appended with the DELIS Tech Report number in square brackets. This indicates the paper was produced within the DELIS project, not some other project.

Contents

1 Method and Approach

In this workpackage we aim to take some of the ideas and techniques produced within other DELIS workpackages (particularly from SP4 and SP5) and apply them to to scenarios / applications that embody plausible engineering constraints. We present here initial work on two application domains - Broadcasting (section 2) and Content Replication (section 3).

The techniques we are attempting to apply have been developed and tested within highly abstracted domains, such as the use of game theoretical abstractions representing interactions or simple fixed graphs representing computer networks. Moreover, constraints and assumptions are often chosen for theoretical convenience or because they relate to assumptions inherited from biological or social theory. But the constraints of engineered systems are often tighter than biological or social systems. For example, engineered systems do not physically evolve and neither do they embody ideal or classical rational behavior.

Our aim therefore is to produce algorithms with the desirable properties of abstract models but also functioning within plausible application-like engineering constraints. This becomes an intermediate stage between theory and application. We have found that extensive computer simulation is currently the only viable way to proceed in this endeavor. Interestingly, we have also found that even minor changes in constraints, that move us closer to useful applications, often require significant experimentation with, and modification of, existing abstract algorithms. Yet, it is possible to import desirable properties.

Another major aim of this work is not just to produce applied algorithms but to identify and develop generic methods. We discuss some ideas on this in the following subsections.

1.1 Model Chains

Our method has been to link a chain of simulation models ranging from abstract models displaying desirable properties to specific application scenarios. A number of intermediate models may lay along the chain. As models become more specific they add new constraints in the form of scenario specifics and / or modified functionality derived from requirements for efficient handing of the scenario. The modifications result from experimentation with simulation models rather than *a priori* deduction from previous models.

We are currently at an intermediate stage with both broadcast and content replication applications. Figure 1 shows the model chains for both. The models presented in the subsequent sections (Broadcast and CacheWorld) are at the level of application domain abstraction, rather than specific implementation level.

1.2 Modeling the Scenario

A key aspect of developing abstract models towards applications is the production of plausible scenarios. That is, test-bed environments that situate the model within a realistic scenario at some level of abstraction. As a model moves from abstract to specific the effort required in producing plausable sceanrios increases significantly. This is because high-level abstractions are no longer viable tests of performance. For example, much of our previous work has relied on game theoretical abstractions which test cooperation between nodes - such as the Prisoner's Dilemma game. Although real world interactions may reflect the logic of the game, the specifics are generally less easily represented. They involve variable loadings, submission and sharing of queries, noise, and less than clear payoff functions or utility measures (if at all).

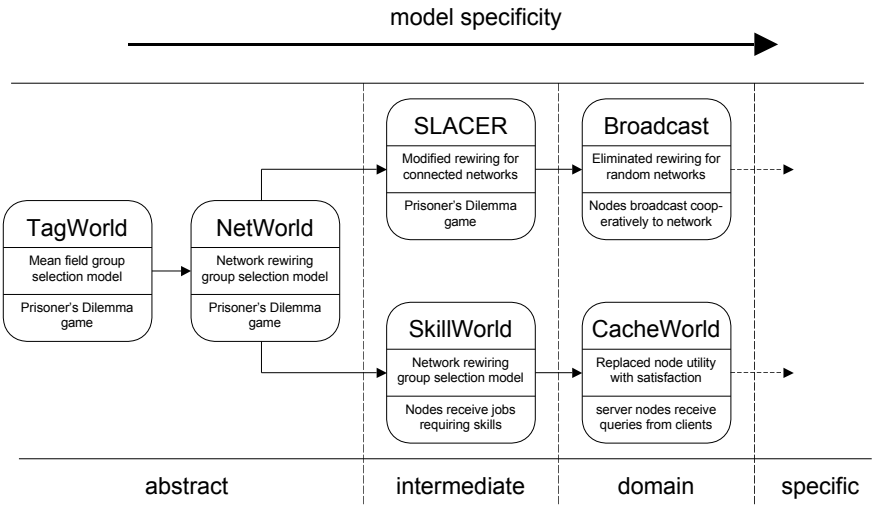


Figure 1: Model chains for the two application domain models included in this report. The more abstract models are to the left, more specific to the right. Although both Broadcast and CacheWorld have a common lineage in TagWorld [9] and NetWord [8] they differ considerably as they are modifications of intermediate models (SLACER [7] and SkillWorld [6]). For each model a brief description plus the scenario used are given. Note that the more abstract models use the Prisoner's Dilemma game as a test.

1.3 The Challenge of Utility

Our initial abstract models are derived from biological and sociological approaches. In both of these approaches models often assume that individual behaviors (or strategies) evolve via reproduction or imitation. This is captured through the notion of an abstract utility or fitness. The idea is that all individuals can be placed along a single dimension of "goodness" which reflects how well the strategies held by those individuals will reproduce relative to others. The idea is that such values represent the outcome of some unmodeled process (i.e. number of offspring in biology or number of others who imitate a behavior in cultural evolution).

When moving to distributed applications some method is required for nodes to calculate a utility value. Also, nodes need to report their utilities honestly if they are to make correct comparisons. However, it is not always clear how, or if, such utilities can be meaningfully calculated by nodes in given scenarios. In addition, nodes may lie about their utilities for individual benefit. We have identified four broad approaches to the problem:

- Test a set of candidate utility functions and select the one giving better performance
- Test the sensitivity of the system to different kinds of malicious, utility lying, behavior
- Dispense with utility altogether and use binary node satisfaction function
- Assume utility represents some out-protocol user activity or perception

We have experimented with such approaches already. In [2] we experimented with various kinds of utility lying behavior in nodes. We found that, for an abstract cooperation game, the system showed some resilience to some kinds of attack but serious vulnerabilities to others. In our initial work on content replication (see section 3) we have moved to a binary satisfaction function, rather than utility, which is not required to be communicated. This potentially offers a solution to the

utility lying problem but more work is necessary. In [5] we discussed how an out-protocol approach might be applicable to a generic "social bootstrapping protocol". Essentially this latter approach requires the input of a utility value from outside the protocol based on perception of performance or quality of service.

1.4 Evolution as Copy and Rewire

We have already produced a highly generic model (NetWorld) that has been applied previously to a file-sharing domain [8]. Interestingly, this same model became the common abstract ancestor model in the chains of both the Broadcast and CacheWorld domain models.

Within NetWorld we applied a generic way to translate biological and cultural mean field evolutionary models into P2P type network models. Since real computer systems do not reproduce psychically and open systems can not rely on the ability to recruit new processors or nodes at will, direct application of evolutionary algorithms into distributed systems is not possible. We have adapted a selection method know as "tournament selection" into a simple on-line runtime copy and rewire process that can be applied in any network in which nodes have access to a random node selection service and can calculate and report utilities. We have discussed this previously elsewhere [8]. However, We believe this is a highly generic approach and could be used to adapt many existing evolutionary models should their properties be desirable in networks.

1.5 Towards Design Patterns

We believe that the abstract models encompass something akin to "design patterns" which could be represented and described in a way that could allow programmers to adapt them for their specific needs. This would involve describing context and conditions of application along with specific examples. Already previous work from the now completed EU BISON [14] project has represented several bio-inspired mechanisms in this format [3]. We have begun to develop a framework for such a pattern based on group selection (see DELIS deliverable D5.4.2) and envisage the development and refinement of further such patterns that we have uncovered - for example, the previously mentioned "evolution as copy and rewire".

1.6 Summary

In this section we have given some of our ideas on the features of a generic method that might be applied in developing abstract biological and social models towards applications. We are far from some kind of prescriptive method but believe we have outlined some of the broad features that such a future method might embody: chains of models, design patterns and utility grounding methods. In the next sections we consider on-going work in two specific example domains.

2 Broadcasting

Many peer-to-peer (P2P) applications require periodic broadcasting of messages from a single node to the entire node population. For example, a node may wish to inform the network of the availability of a new service or resource, or a node may want to query the entire network for a particular file or service. In unstructured P2P networks a simple solution is to use a flood-filling (FF) method. In the FF approach each node passes the message to all its neighbors in the network when it first receives it. However, this approach, though robust to high node turnover and failure, is highly inefficient because many more messages are sent than is required. Additionally, this approach is vulnerable to "free-riding" nodes that may receive messages but not waste their own resources (bandwidth) passing them to neighbors. Here we present a simple modified FF method that increases efficiency and reduces

the incentives to free-ride while retaining robustness and simplicity. The method presented makes use of an evolutionary approach in which nodes copy strategies of others. Interestingly, this appears to self-organize the network toward a critical threshold of the network. This appears to be a novel process which may have application beyond broadcasting.

This novel approach arrived at by initially applying the previously produced SLACER protocol [7]. and then tuning the rewire probability parameter. After much experimentation we found that a value of zero actually gave the best results in simulation. Interestingly, this means that all rewiring is actually disabled. Hence, what we are left with is a simpler protocol that operates over a random network utilizing a novel group formation process. The rest of this section summarizes the work described in [1]

2.1 Introduction

Many P2P applications require a general broadcasting service that will propagate a message from any node to all other nodes in the network. For example, file-sharing systems such as gnutella[2] require queries from nodes to be passed over the entire network. P2P networks vary in their topologies, some maintain fixed structures while others follow unstructured and highly dynamic schemes.

Here we examine broadcasting in unstructured and possibly dynamic P2P overlay networks. Overlay networks are logical topologies constructed over existing infrastructures (such as the Internet). Each node maintains some small number of logical links to a set of neighbor nodes. As an illustration an Internet based P2P overlay may store a set of IP addresses in each node indicating which nodes are connected and can communicate directly.

A simple way of implementing a broadcast function is for each node to pass all new broadcast messages it receives to all of the other nodes it connects to (neighbors). In this way a message initiated from any node will eventually reach all other connected nodes in the network. This is termed a flood-fill (FF) approach and although simple and robust can be highly costly in terms of messages required to be passed: FF will send L messages (where L is the total number of links in the network) assuming the network is connected.

The optimal broadcasting possibility in terms of messages sent, is for the message to follow a spanning tree in which each node only passes the message to those neighbors that have not already received the message. This approach requires only N messages (where N is the total number of nodes in the network).

It can be seen that, given a static network, a spanning tree can be constructed for a given source node via an initial FF broadcast in which node information concerning message spread is retained. Hence for certain kinds of relatively static networks where some small number of source nodes will initiate broadcasts this approach can be used. However, in many P2P applications networks are highly dynamic and any node in the network may need to initiate a broadcast.

A further issue in many P2P systems is that they are open in nature which means that there is no central administrative control that can ensure all nodes follow a protocol exactly. Anyone can modify and release peer clients that may behave in selfish or malicious ways. In the context of broadcasting, a node may happily receive messages but not pass them on to its neighbors because this might incur a cost such as bandwidth or power consumption. Given these factors many P2P designers opt for a pure FF approach.

We formulated a different approach in which nodes dynamically decide if to pass on messages based on a simple adaptive evolutionary protocol. Using simulations we found that such an approach, though motivated by self-interest in the nodes, produced reasonable broadcasting performance with a total message cost significantly less than that required for the FF approach. Interestingly, we found that our simple protocol appears to evolve the network towards a critical threshold (k) for the

[2] see http://www.gnutella.com

network that could be related to site percolation theory.

In previous works [12] broadcasting applications have been optimized via *a priori* analysis of the site-percolation critical threshold. In those studies, the threshold value is then built into the protocol. The approach given here differs from these approaches in that the network itself self-organizes around the critical threshold for a given network. In addition the nodes behave in a myopic and selfish way - copying the behavior of other nodes that obtain higher utility.

2.2 Broadcast Scenario

For simulation purposes we assumed that the network forms a random graph topology. We tested both static and highly dynamic graphs. Each node in the network takes one of two states or strategies (PASS or DROP). A PASS node always passes a newly received broadcast message to all its neighbors (excluding the neighbor it received it from). A DROP node is a "sink" and does not pass any message it receives.

2.2.1 A Broadcasting Game

We formulated the simulation in the form of a "broadcast game" in which nodes were awarded a payoff (or utility) based on the outcome of each broadcast event. A broadcast event involves randomly choosing a node from which a broadcast message is initiated. The node passes a message to each of its neighbors. In the next time-step each neighbor then decides if to pass the message to it's neighbors based on it's current strategy (either PASS or DROP). This process continues until no more messages can be sent.

We assume that nodes wish to receive broadcast messages but that nodes have some incentive not to pass messages on. Hence we assume there is a benefit for receiving a message (B) and some cost for sending a message (C). We assume that $B > C$.

After a broadcast event each node in the network will either receive a utility award of B, if it received but did not pass the message, $B - C$ if it received and passed the message or zero if it did not receive the message.

2.2.2 Simulation Time Cycle

For the purposes of simulation a time cycle represents a period during which all nodes are activated (in a random order). When a node is activated it checks it's input buffer for a new message. If a message is found then the node either forwards it to all its neighbors (excluding the node from which the message was received) if it stores a PASS strategy or does nothing if it stores a DROP strategy. The utility of the node can then be updated as described above.

After message handling the strategy update action is performed with probability P. Another node is selected randomly from the entire population[3] and a utility comparison is made. If it has a higher utility then it's strategy is copied (or replicated). If a replication is made then mutation is applied with some small probability m to the strategy. Mutation involves flipping the strategy. In this way, nodes receiving higher utility will tend spread their state to nodes with lower utility via replication but mutation events allow nodes to spontaneously switch strategies. Replication step pseudocode is illustrated in Figure 2.

2.2.3 Parameter Values

We used a network of size $N = 4000$, with each node having 20 neighbor links. The payoffs were $B = 1$ and $C = 0.25$. Replication probability was set to $P = 0.2$ and mutation rate $m = 0.005$. Broadcast

[3]A random peer sampling service is required. In our implementation NEWSCAST[10] has been used.

```
j ← GetRandomNode()                  // random node picked for comparison
if ( Utility_j > Utility_i ) then    // utility comparison
  S_i ← S_j                          // strategy copy
  with probability m change S_i      // strategy mutation
  Utility_i ← 0

        S_i ∈{PASS, DROP} denotes behavior (strategy) of node i
```

Figure 2: Replication step pseudocode. This is a simplified form of the previous SLACER protocol. Here only strategy is copied. Links between nodes are not copied.

events were initiated every $E = 20$ cycles[4] Simulations were run on both fixed and dynamic networks where node were periodically removed and added in order to simulate *churning* phenomenon typical of P2P networks.

2.3 Simulation Results

To analyze the performance of the system three measures were calculated:

- Passing nodes (PASS): Proportion of nodes adopting the PASS strategy.
- Message Count (MC): Number of copies sent for any single message.
- Number of Receptions (NR): Proportion of nodes receiving a message.

Obviously PASS and MC are related (the more nodes forward received messages, the more messages copies are sent). On the other hand to obtain a high value of NR it is not necessarily required to have a high PASS value. A high PASS value translates into high redundancy in terms of many messages being sent to nodes that have already received them.

Our objective is to have the highest possible NR ($NR = 1$, meaning a message is seen by all nodes) while keeping PASS and MC values as low as possible. In other words we want as many nodes as possible to be able to receive the broadcasted messages while minimizing the number of message copies sent.

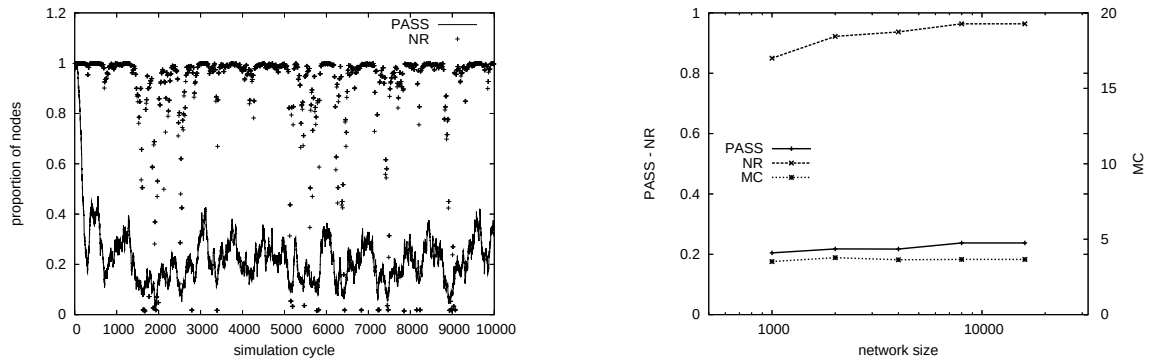
In the rest of this section simulation results concerning both fixed and dynamic networks of different sizes are shown and discussed. In the following section we propose an interpretation consistent with the results based on a critical threshold.

2.3.1 Fixed Topology

To generate the random topology the network is initially composed by N nodes and no edges at all. To define edges one node (say i) at a time is picked and 20 links (set as the viewsize limit) (i, j) to randomly picked nodes j are added. If a node j has full view then no link is added. This method leads to a random topology where most of the nodes have a full view while some nodes (the last ones picked for adding links) are not able to fill their view due to the difficulty to find random nodes whose view hasn't been completely filled yet, since the network is constructed adding links sequentially.

Figure 3(a) shows the typical trend of a single run of the algorithm on a 4000-node network. Node strategies are initially set to 100% PASS, hence flood fill is performed, a new message is generated every 20 simulation cycles and broadcasted starting from a randomly picked node. Each node probabilistically tries to evolve its strategy at each cycle with probability 0.25.

[4]This gap in broadcasts was made for purposes of analysis. The model allows for broadcasts to sent from any node at any time because nodes have internal message buffers.



(a) Typical single run

(b) For different network sizes

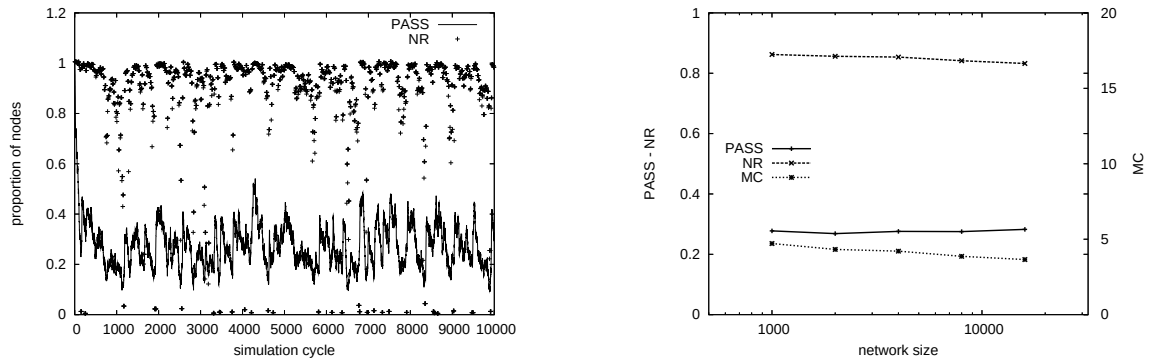
Figure 3: Results for a static network. (a) shows a single run on a 4000-node network. PASS shows the proportion of nodes adopting the PASS strategy. NR shows the proportion of nodes actually receiving a message. MC shows the average number of message copies sent per node. A new message is generated every 20 cycles. It is interesting to notice that PASS initially decreases, but stops dropping as soon as NR value degrades. (b) shows results on networks of different sizes. Notice here we have a reverse scaling effect on NR - larger networks perform better.

At the beginning of the simulation the PASS value, initially set at 100%, immediately starts to drop while NR remains very high. This indicates that not all nodes need to follow the PASS strategy for the message to propagate over the entire network. However eventually PASS gets too low to make it possible for a message to be spread throughout the whole network, resulting in a drop of the NR value (around cycle 1500 in the example shown in figure 3(a)). Notice how at this stage the PASS value stops dropping and starts oscillating.

It appears that the system lowers PASS as long as there is some level of redundancy. When the PASS strategy falls below the critical threshold required for network wide message propagation PASS oscillates around that value. Throughout the simulation there are sudden dips (or dropouts) in NR indicating messages that reach a very low number of nodes, this could happen by chance if a node surrounded by DROP nodes is chosen as a source for a broadcast event, or as a result of a PASS value too low to sustain good message spreading. The system reacts well in both cases, in fact while occasional dropouts due to an *unfortunate* source node selection do not affect the PASS trend, continuous dropouts due to a too low PASS value cause PASS to increase, bringing NR back to a high value as a consequence.

In figure 3(b) algorithm performances related to different network sizes ranging from $N = 1000$ to $N = 16000$ nodes are shown. Each point is the average value of 5000 distinct broadcasts (10 different runs of 10000 cycles).

Looking at the NR value, it is high (above 0.8) for every network sizes and reverse scales with the size of the network, approaching 100% spreading in 16000-node networks. Although some nodes fail to receive messages it is important to note that because node strategies are continuously changing it is unlikely that the same nodes repeatedly fail to receive messages. Moreover let us notice that NR gets higher with bigger networks, such reverse scaling is a desired property in P2P network environments since P2P networks are usually extremely large and possibly growing networks. The system also scales for the PASS and MC values obtained because their values remain constant the range of network sizes.



(a) Typical single run

(b) For different network sizes

Figure 4: Results for a dynamic network. (a) shows a single run on a 4000-node network. PASS shows the proportion of nodes adopting the PASS strategy. NR shows the proportion of nodes actually receiving a message. MC shows the average number of message copies sent per node. A new message is generated every 20 cycles. (b) shows results on networks of different sizes. Notice that here we get some degradation in NR as network size increases. Hence there is a scaling cost.

PASS and MC values are related and can be used to give an estimation of the average node degree. Since each time a node passes a message it is forwarded through all of its links, the average degree k could be estimated as $k = \frac{PASS}{MC}$. As an example, we know that for construction node views tend to be filled, so that most of the nodes have degree 20, and in general nodes have high degree. Applying the formula to PASS and MC values in figure 3(b) a value slightly lower than 20 is always obtained.

2.3.2 Dynamic Networks or Churning

To evaluate network performance in presence of churning (dynamicity introduced by nodes continuously leaving and entering the system) we ran experiments with the same setting described in the static case and in addition at each cycle we removed 1% of the network nodes randomly chosen, and reintroduced the same quantity of nodes linked to 20 (viewsize) random preexisting nodes initially playing a random strategy uniformly chosen between PASS and DROP.

As can be seen in figure 4(a) the effect of churn is twofold: on one hand PASS oscillates around higher values than in the static case, on the other hand NR drops more often.

The high PASS value is a result of the entering nodes using strategy PASS with probability 0.5 (with the system that usually stabilizes at a lower value). Concerning the lower NR it could be seen, on the other hand, as a consequence of nodes leaving the network because when a node is removed its incoming message queue containing messages to be forwarded at the next cycle is implicitly removed.

Differently from the static case, as can be seen in figure 4(b) even though NR remains high, in dynamic networks NR decreases when the network grows. This could be an effect of the relative growth of churning rate with respect to the message spreading speed. While churning rate is always set to 1% relative to the network size, message spreading speed is bounded by the network topology, specifically by the maximum viewsize which is fixed independently from network size. We note that the churn rate we adopted to test the network is extremely high: basically 1% of the node enter and leave the network in the time space needed by a node to send a simple message to a neighbor (milliseconds order of magnitude). Possibly, the network would scale better with lower churn rates.

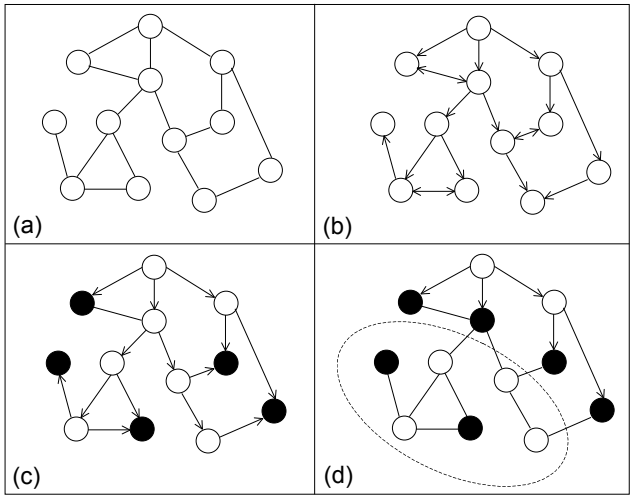


Figure 5: Broadcasting on a simple graph with PASS and DROP nodes. (a) shows a graph representing the network. (b) shows the route messages take through the network assuming that the uppermost node initiates a broadcast. (c) shows the same graph with some DROP nodes in black, notice that in this case all nodes still receive the message. (d) shows a situation in which a portion of the nodes (inside the dotted region) do not receive the message.

2.4 Critical Threshold Process

The results presented in the previous section may initially appear to represent a puzzle. Given that nodes always have an incentive to use a DROP strategy why would not all nodes end-up using a DROP strategy? Also, why does the number of PASS nodes oscillate around a value that appears to be a critical threshold?

We believe that the process operates via a kind of self-organizing evolutionary process around the critical threshold for the network. We summarize the process in the following way:

- When PASS is high there are many routes between nodes that messages can take
- Nodes can can adopt DROP behaviors without affecting NR
- But when PASS falls below some critical threshold (k) NR goes low and many nodes get zero utility because they dont receive a message
- Both PASS and DROP nodes receiving the message are equally likely to be copied by nodes that dont get the message
- Hence the PASS proportion found in the cluster of nodes receiving the message would tend to be reproduced by zero utility nodes
- However, the PASS nodes receiving the message will tend to copy the DROP behaviour of nodes receiving the message
- So a highly dynamic process of cluster connection and disconnection ensues around a critical value (k)

In summary, when PASS falls below a critical threshold k then nodes can be divided into a connected cluster around the source node that receive the message and those in disconnected clusters

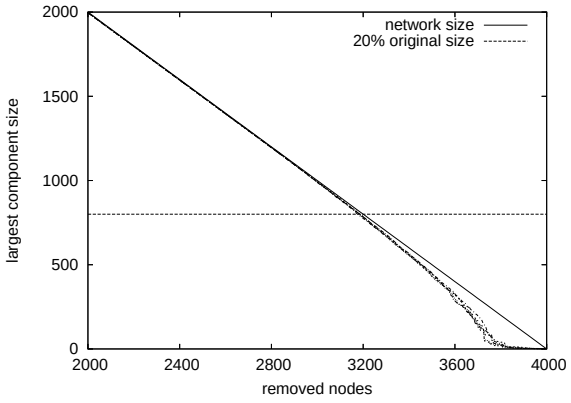


Figure 6: 4000-node network connection. At each cycle 0.1% random nodes are removed. Diagonal lines represent actual network size, while the dotted lines represents the size of the biggest component in different runs. 20% of the initial size is indicated and corresponds to the point where in all the simulation performed and shown the bigger component gets lower than the network size, suggesting network partitioning.

that don't receive the message. Given the utility payoffs, the disconnected nodes will tend to reproduce the proportion of PASS from the connected cluster and this proportion should be around the critical threshold at the point the network disconnects. Figure 5 illustrates this notion with a small graph (a) and the route messages would take if sourced from the uppermost node if all nodes used a PASS strategy (b). Note that in (c) a number of nodes use a DROP strategy but all nodes still get the message. However, in (d) a region of the network does not receive the message.

2.4.1 Simulation Experiment

To check if the value of $PASS = 0.2$ corresponds to a critical threshold k, under which the network becomes disconnected, we tried a very simple simulation experiment. Many fixed random networks were generated with the algorithm described in section 2.3.1,then 0.1% of nodes are removed progressively (a removed node corresponds to DROP behaviour) and the size of the biggest component, analogous to the part of the network able to receive the messages, was calculated.

As shown in figure 6 these experimental results suggest that $k = 0.2$ and hence agree with our findings in the broadcast scenario and increase our confidence on the ability of the algorithm to tune itself to the critical threshold value.

2.4.2 Possible Link to Percolation Theory

The value reached by PASS is the lower possible value in which the network is still able to send messages. To put it another way it is the minimum number of nodes (and relative links) needed to be active for the network not to be partitioned. This description is very similar to *percolation threshold*.

In percolation theory each site in a graph (node in a network) could be either active or inactive. Percolation threshold of a graph is a phase transition point indicating the minimum proportion of nodes that should be active to ensure that they form a unique connected cluster. While for simple

topologies such as square lattices exact threshold values are known, for more complex topologies many analytical models, as the ones in [4, 11], exist.

Unfortunately in our system we are not only interested in PASS nodes being part of a single cluster, but we even want messages to be spread throughout the network, in order to obtain this we also need each DROP node to be connected to at least one PASS node. This additional constraint makes the problem we want to solve similar to the *vertex cover* problem. A vertex covering for a given graph G is defined as the set of vertices V so that every edge of G is incident to at least one vertex in V. Finding the minimum vertex cover in a graph is an NP-complete problem.

The threshold found through simulation in our system seems to share properties both with vertex cover in the sense that we want each node (even DROP ones) to receive messages, hence to be reached by at least one active link, and with percolation threshold for the reason that PASS nodes should be connected so to play the role of a *backbone* through which messages are spread throughout the network.

Percolation threshold and the vertex cover in random networks are hard problems, but the possible relations between these two well-known and studied problems and our simulated system increases our confidence on the possibility to give an analytical solution of our broadcast model and on the goodness of the threshold value adaptively reached through a simple evolutionary scheme.

2.5 Summary

In this paper we have shown how a simple algorithm running in the nodes (a P2P protocol) can self-organize a network around a critical threshold using an evolutionary approach within a broadcasting scenario. Even though nodes have an incentive in the scenario to drop messages the systems keeps the number of nodes passing high enough such that most nodes receive messages. This approach is scalable and robust to churn (highly dynamic networks).

The process which keeps the number of passing nodes from dropping permanently below the critical threshold appears novel and interesting since it appears to exploit a global property (the network becoming disconnected) which feeds-back to the local behavior (evolution based on individual utility) to give a desirable self-tuning result.

There are still open issues which need to be addressed in order to develop our approach towards a specific P2P protocol that could be considered a model for implementation.

We have formulated the broadcast problem as game and use utility payoffs. As we have discussed, in section 1.3 above, This introduces all the concomitant utility issues. How exactly, in a more specific scenario, would utility be calculated? We would have to assume that the broadcast messages contain some kind of useful information that nodes need to receive in order to keep their utility high - currently we have not modeled this but just assumed it. Also, currently there is no utility benefit (or hence incentive) for nodes to initiate a broadcast and this may be unrealistic.

As in previous work we assume honest reporting of strategies and utilities and this potentially means the protocol could be vulnerable to malicious attacks based on lying. Also, although the number of messages sent for broadcasting appears to be lower than a flood-fill approach, we have not factored in the message overheads required for communicating strategies and utilities, this is an additional costs that needs to be considered if a true comparison is to be made. This could undermine the advantages.

The technique we proposed for optimized broadcast in P2P networks exhibit many interesting properties, such as self-organized criticality in adapting the number of passing nodes on a limit value needed to spread messages throughout the whole network, good scalability and resilience to churn. The possibility to relate our approach to well-known and established problems in percolation theory or connected cover sets could make it possible to perform further analytical investigation for the the design of additional algorithm features and optimization.

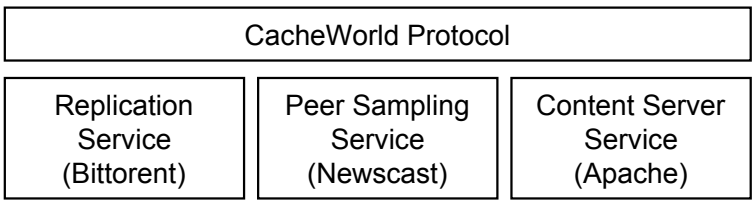


Figure 7: CacheWorld depends on three existing services to operate: a content server service; a content replication service and a peer sampling service. In brackets we give an example of an existing service / protocol which might be applicable.

Although our interest is from a P2P engineering perspective, perhaps the evolutionary mechanism demonstrated could be applicable to physical phenomena observable in the world.

3 Content Replication

As demand for information increases, centralized servers become a bottleneck. Content providers, system administrators (on behalf of their users - either in the enterprise or at the ISP) or end users themselves cope with the problem of performance by distributing replicas of web content to machines scattered throughout the network. The replicas then respond to local client requests, reducing the load on the central server. Load which is distributed to the nodes which cooperate includes:

- communication bandwidth, for transmitting the data to the requesting content,
- storage used for hosting the replica, and
- CPU resources for query processing.

Replica Management in this context refers to the problem of deciding how many replicas of each file to distribute, and where to place them. Enough replicas should exist to handle the cumulative demand for each file. With too few replicas, servers become overloaded, and clients see reduced performance. Conversely, extra replicas waste bandwidth and storage that could be reassigned to other files, as well as the money spent to rent, power, and cool the host machine.

Our aim is to research mechanisms which can be engineered into peer-to-peer data replication protocols, that allow the emergence of policies which are a good tradeoff between maximizing the benefit of individual nodes and the group ones.

Our approach uses a simple protocol that dynamically self-organises clusters of servers that mutually replicate content and service redirected queries in order to share load. To do this we have adapted protocols from previous abstract models which tested behavior using simple interaction games (such as the Prisoner's Dilemma) to capture the potential contradictions between individual and social goals.

Here we present initial simulation work and discuss what needs to be done to refine the model towards a plausible implementation.

3.1 The CacheWorld Scenario

We assume a population of N server nodes which form a P2P overlay network. In addition to being part of the overlay, each node functions as a server responding to requests (queries) which come from clients outside of the overlay network (clients). An example could be that each node is a web server

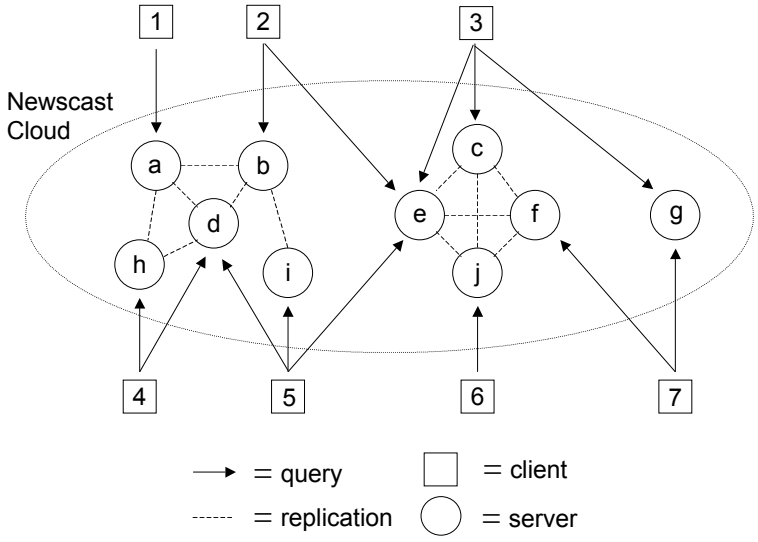


Figure 8: Schematic of the CacheWorld Scenario. Sever nodes connected in a P2P overlay network are required to service queries for content from clients that are outside the overlay. All server nodes are within a Newscast cloud. Each server node is linked to some other nodes (neighbors) which it replicates content from and vice versa. If a node becomes overloaded with queries from clients it will redirect the query to a neighbor. A concrete example is to view server nodes as web servers and clients as web browsers.

with the overlay linking the servers and clients being web browsers on remote machines requesting content from the servers.

We assume each node always stores one copy of its own content item which it serves to clients (e.g. a website) and that it has additional storage space to store k replicated content items from other nodes which it can also serve. The overlay network directly links nodes bidirectionally if they mutually replicate each others content item. In our web site example this would mean that when two nodes are linked each holds a copy of the others site. Directly connected nodes are said to be "neighbors".

Figure 8 shows a schematic with server nodes receiving requests from clients and linking to other nodes for mutual replication. Below we discuss the main features of the system.

3.1.1 Query handling

Over a given time period nodes receive some number of queries (load) from clients to serve their own content item. Each node has a maximum capacity, c, which specifies the maximum number of queries it can serve in the given time period.

If the load exceeds the capacity for a given node then some queries can not be satisfied by the node and the node is said to be "overloaded". When a node becomes overloaded it redirects queries it receives to a randomly selected neighbor. If the neighbor is not itself overloaded it will serve the query from its local replicated content copy, otherwise it will ignore the query. Hence, each node calls on randomly selected neighbors to service queries when it is overloaded. The essential idea therefore is that overloaded nodes will have neighbors that are not overloaded and can serve queries they can not. Every query served by a node, either directly or from a redirection, uses capacity. It is therefore possible that a node can become overloaded due to serving redirected requests.

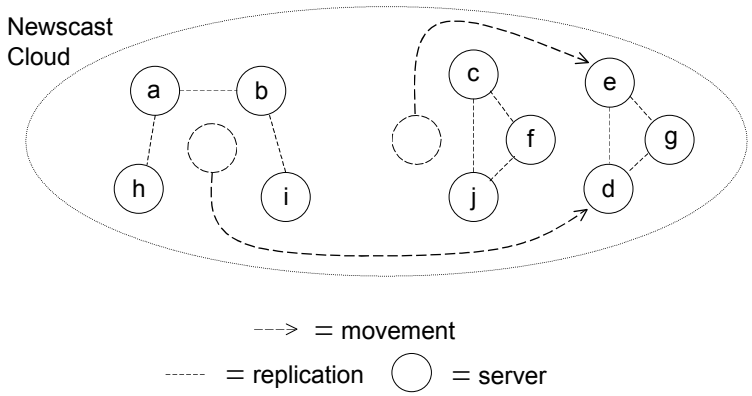


Figure 9: Schematic of node movement the CacheWorld Scenario. Here two nodes which have become unsatisfied move by selecting new neighbors randomly. This moves would only be accepted by the new neighbors if they also were not satisfied.

3.1.2 Satisfaction and movement

Nodes maintain a periodically updated estimate of the proportion of queries for its own content that are actually served (ps). A node is said to be satisfied when $ps >= t$, where t is some threshold value. Periodically, and asynchronously, nodes change their neighbors (move) in the overlay network if they are not satisfied.

Nodes select from one of two kinds of move operation: either to 1) "go it alone", or, 2) "find new neighbors". The first option is selected if the node determines that it could have been satisfied if it had not had any neighbors at all. That is, if its capacity is at least equal to its individual load. The second move option is selected otherwise.

When a node moves it first drops all its current neighbor links. If it has decided to "go it alone" then the move operation is complete. In the case of "find new neighbors" the following additional steps are executed, the node: a) selects a random node, j, from the population; b) links to the neighbors of node j; c) links to j. If at any time k is exceeded then a randomly selected existing link is dropped. Figure 9 shows an example of two nodes moving to find new neighbors in the overlay. Figure 10 gives outline pseudocode for the protocol.

As stated previously, It is assumed that when a node links to another node they mutually replicate each others content items. Hence movement resulting in linking to new neighbors implies a cost due to the replication process. Currently we assume this is negligible, which would be unrealistic in many scenarios where content items were large and / or bandwidth between server nodes was low. However, with large k a swarming would be possible (since all neighbors of a node would store replicas).

3.1.3 Random node sampling

In addition to being members of the replication overlay network each node is also a member of a Newscast Cloud. This provides all nodes with the ability to locate a random node from the entire population which is required for node movement. Newscast itself operates by maintaining its own overlay network which supports a random and connected topology. Newscast has been covered elsewhere, and we do not go into details here [10]. Suffice to say that Newscast provides a suitable random node sampling service which is required. Other random node sampling protocols have been proposed and they could equally well be used.

```
Passive thread

on receiving a query q, node i:
    if not overloaded
        service q directly
    else if neighbors > 0 and q is not already a
    redirected query
        j ← selectRandomNeighbor()
        redirect q to j
    end if
```

```
Active thread

periodically each node i:
    if not satisfied
        drop all neighbor links
        if Ci < directly received queries
            j ← selectRandomPeer();
            link to j's neighbors and j
        end if
    end if
```

Figure 10: Outline pseudocode for the CacheWorld protocol. The passive thread is activated when a node receives a query either directly from a client or as a redirection from another node. The active thread is activated periodically approximately once per load cycle. The value Ci represents the capacity of node i. This is the maximum number of queries it can serve per load cycle.

3.2 Simulation Specifics

A simulation model was implemented within the *Peersim* [13] system, an open source P2P systems simulator platform. Our initial experiments have been performed with a small number of nodes $N = 50$. However, we found our results to be broadly scalable. We ran tests up to $N = 5000$. For our initial experiments we set the maximum number of links a node can have to $k = 3$. This means each node can replicate up to three other nodes' content at any given time.

3.2.1 Load Cycle, Load Profile and Capacity Profile

Peersim divides time into cycles (we call these *Peersim Cycles*). Some number of Peersim cycles constitutes what we term a *Load Cycle*. In each Peersim cycle each node i is fired, in random order, and may receive a client query. If a query is received, it will be answered directly if the node is not currently overloaded. If the node is overloaded it will redirect the query to a neighbor node if it has one. The neighbor node will then answer the query or, if it is also overloaded, will ignore it.

The number of queries (load) given to each node $1..N$ in one load cycle constitutes what we call the *Load Profile L*. For a node i, Li specifies the load for that node. If L specifies the same value for each node then all receive the same number of queries and their load is equal. Variation in loads between nodes indicates that some nodes receive more queries than others. For our initial experiments we considered a very simple load profile in which half the nodes we given $Li = 5$ and the other half $Li = 15$. This means that half the nodes are twice as likely to receive a query as the other half and hence will have twice the load.

Each node also stores an individual capacity value, Ci, that specifies the total number of queries it can serve over a Load Cycle. The set of all Ci capacity values for all nodes $1..N$ constitutes what we call the *Capacity Profile C*. Hence the sum of all Ci values constitutes the total system capacity, TC, which is the maximum number of queries the system as a whole can serve in one load cycle.

For the purposes of analysis we wish to determine how the system behaves when it is at maximum load. To achieve this we set the total load profile to be equal to the total system capacity. In these initial experiments we set the capacity profile such that all $Ci = 10$, hence $TC = 10N = 500$. This is equal to the total load.

3.2.2 Movement and satisfaction

At the end of a load cycle each unsatisfied node considers moving in the overlay with probability $mp = 0.1$. A node is not satisfied if the proportion of queries, it directly received from clients, that were eventually served, ps, is below threshold t. Here we set $t = 1$ for all nodes, so a node is only satisfied if *all of its queries are served*.

3.2.3 Performance measures collected

In all cases experimental results were produced by collecting results from 10 independent simulation runs that ran for 100 load cycles each. To asses performance we recorded three measures, $Q, SandM$, that were calculated as averages over all runs for the last 50 cycles. In earlier cycles performance varies due to initially high levels of node movement. Q gives the proportion of queries actually served as proportion of all submitted; S gives the proportion of nodes satisfied and M the amount of node movement - the proportion of nodes that actually move per load cycle.

3.2.4 Random peer sampling service

As stated previously the system maintains a Newscast Cloud. The Newscast protocol is already implemented within the Peersim system as a protocol module and we used this to maintain the population level random peer sampling service. Each node calls it's Newscast protocol when fired in the Peersim cycle. When a node moves and requires a random node it makes a single call to the Newscast $selectRandomPeer()$ interface, which returns a link to a random node from the population. There is no other interaction between Newscast and CacheWorld protocols.

3.2.5 Parameters and measures summary

The parameters used in experiments were: Network size $N = 50$; Maximum degree $k = 1$; Initial topology: *random graph*; Capacity profile: all nodes have capacity $C = 10$; Load profile: half nodes 15, other half 5; Load cycle: $N \times C = 500$; Satisfaction threshold: $t = 1$; Movement rate: $mp = 0.1$.

The performance measures collected from the experiments were: Proportion of all queries served: Q; Proportion of satisfied nodes: S; Proportion of node movement: M.

3.3 Some Initial Results

We present results from three different experiments (a-c). For experiment (a), as a based line, we ran the simulation with all CacheWorld services turned-off. There are no links between nodes, no query redirection and no movement. Nodes therefore can only answer queries directly passed to them by clients and can only answer a maximum of Ci such queries. Given our load and capacity profiles discussed above it is obvious that we would expect, on average, Q = 0.75, S = 0.5 and, obviously, M = 0 (since there is no movement possible). This is what we found and these results are shown in figure '11 under the columns labeled experiment (a). The variance in results was negligible and are not shown.

In experiment (b) we initialized the overlay network to a random topology with $k = 1$ for all nodes but turned on only query passing. This means nodes are effectively randomly paired. We did not allow node movement. Hence the overlay topology was fixed and unchanging with neighbors assigned randomly and fixed for the duration of the experiments. This gives a further base line showing how well a purely random approach performs. The results are shown in figure 11. As might be expected there is significant improvement because nodes have the chance to now share load. We found Q = 0.84, S = 0.58 and again M = 0 (since no movement is possible).

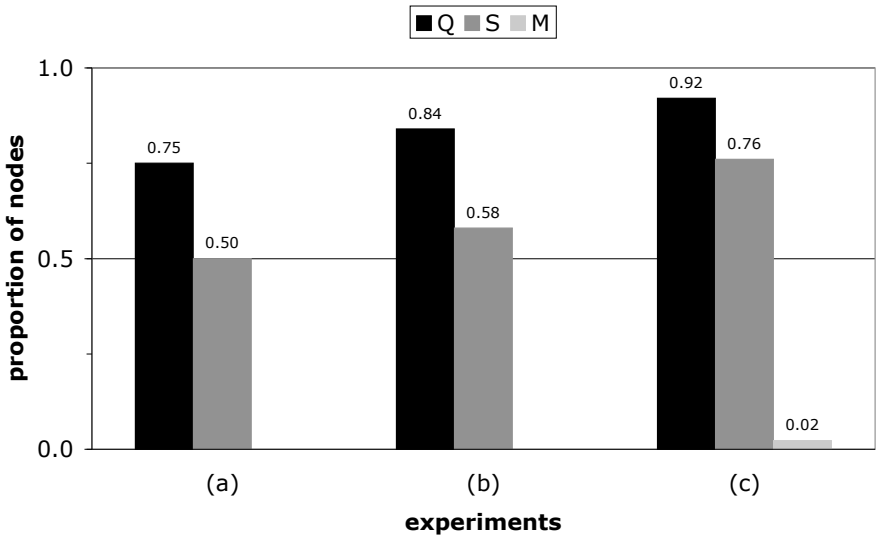


Figure 11: Results from three experiments (a-c). In (a) the CacheWorld service is switched off. In (b) node movement is switched off. In (c) the CacheWorld protocol is fully active. Q shows proportion of all queries answered, S shows proportion of nodes satisfied, M gives proportion of nodes that move each load cycle. Results show averages over 10 independent runs for the last 50 load cycles. Variances were negligible and are not shown.

Finally in experiment (c) we turn on the full CacheWorld protocol allowing movement by unsatisfied nodes. The results are shown, again, in figure 11. We see improved, though far from optimal, performance. We found Q = 0.92, S = 0.76 and M = 0.02. Again variances were negligible.

Movement was roughly constant over load cycles, equating to an average of 1.15 node moves per load cycle. This means that nodes did not settle into a fixed topology but continued to move. Obviously, for a settled topology to emerge, in the given scenario, all nodes must be satisfied and this will only occur when all queries are served. Although such an optimal state is possible, for the given load and capacity profiles, the system does not find it.

The main result here is that turning on the CacheWorld improves total queries served, Q, by 17%, keeping 76% of nodes satisfied and resulting in only a small amount of node movement and hence replication effort. However, this is a very simple loading scenario in which we might hope for better performance.

3.4 Discussion of Results

The idea behind CacheWorld is that the movement of nodes in the overlay network leads to the formation of components of highly clustered peers that replicate each others content. Elsewhere we have called similar formations *Tribes* [7]. A stable tribe is one that satisfies all of its members otherwise they will move away over time.

In our initial experiments we have considered loads where relatively large numbers of nodes are either under- or over-loaded and we have limited each node to only one link (k=1). In these cases stable pairs of nodes emerge. But a small proportion of nodes fail to find a stable partner due to the movement heuristic we have used: satisfied nodes reject new connections. This means under-loaded nodes will never accept new connections. Yet in other experiments, not sown here, we found that always allowing connections reduced performance further due to the consequent addition of noise, which disrupts stable pairings.

More generally, over time, stable clusters would comprise nodes with complementary loads (i.e. loads that tended to be negatively correlated). This may be plausible for real world applications since demand for content often follows temporal daily patterns and clustered peers could be serving content to different time zones effectively supplementing capacity for day time load peaks in one time zone, say, with spare overnight capacity from another. In some sense if we interpreted a load cycle in our model as a half day (12 hour period) then we capture some aspect of this. But in our current experiments the load profile is fixed not dynamic. To capture this notion properly in our model we would need a more sophisticated representation of the load profile and load cycle. The load profile could be made dynamic over load cycles. In this case the load profile would not be a fixed set but a matrix of loads with time and node dimensions. In this case it would be of significant benefit to find real data from servers to set such loads.

3.5 Summary

The model presented here is a first step towards a self-organised and fully distributed mechanism for coordinating content replication between servers to improve overall performance, in terms of serving queries. We have presented very initial results with what we consider to be the simplest possible scenario that could be claimed to capture some of the salient features of Replica Management.

3.5.1 Open Issues and future work

However, a number of issues need to be addressed if this approach is to be moved towards realistic plausibility. For example, we have not tested or refined the system under realistic query loads. In its current form excessive loads on individual nodes could lead to very inefficient outcomes due to "thrashing" as the overloaded nodes constantly move in the network looking for satisfactory neighbors which they never find. We also need to experiment with variants of the movement heuristic to find more efficient approaches. We consider that our approach may benefit from some kind of a simulated annealing technique where movement follows a progressive damping or "cooling", this is on-going work.

We have not tested the system performance when we introduce *free rider* nodes that would not replicate, or serve queries, from others. Based on previous work with similar protocols we would expect some robustness to this kind of behavior [6]. In addition our use of satisfaction rather than utility would give added robustness to cheating based on utility lying [2]. We have not considered how our system would repel a "Sybil Attack" in which a single node connected to a large set of nodes in the population and exploited them without returning any benefits. All these aspects need to be tested.

The current model is, for the sake of simplicity, ignoring the cost of replication, which is probably the highest priority next step in our research. Also, it is not considering the rate of 'obsolence' or update of a content item. When deciding whether to change his own replication strategy or to keep going with same neighbors, these two factors play together and are very relevant to the choice. Cost of replication (especially for large files) is likely a motivation for setting up stable relations among peers (since changing would imply high data transfer costs). On the other hand, obsolescence of content which may become invalid partially mitigates this (since refreshing content which is 'expired' should imply the data transfer cost even in case we do not change neighbor). For several scenarios e.g. music, video, software packages we believe the cost of replication however will tend to dominate. We expect that including cost of replication in the model should improve cohesion within the tribe.

Another desirable extension would be to allow nodes to be neighbors for some content object, not for a whole 'website'. This would allow us for example to use different replication strategies for different type of content (e.g. large vs. small files), to facilitate movement to a new neighbor(e.g. A replicates its content on B up to some point in time, then prefers to move to C for new content but

keeps B for content objects already on B - avoids the cost of replication).

A further step should be to devise mechanisms for optimal replica placement. We want to look at approaches which can produce self-organizing, emergent behavior in placing the content replica in a geographically optimum way based on the access patterns to data from clients.

Currently, for simplicity,the model assumes liked nodes will mutually replicate all of each other's content. We also assume bidirectional links only. Relaxing this constraint would allow for directed links (i.e. content may be replicated from one node to another without a reciprocal link).

3.5.2 Viability for application to real systems

Several Replica Management systems already exist and are in use. We need to assess these and compare with our approach.

Most existing systems (e.g. Squirrel [15], Coral [16], Globule [18]), which use replication, do not have specific mechanisms for adjusting the cooperation among nodes. For example, for balancing global policies (i.e. the goals of the community) against local (i.e. that of individual nodes/ agents) policies. The policies in such systems are fixed by design, after using simulation to understand good trade-offs in the protocol design phases (example policies include: how much storage devote to storing files from other nodes, ratio of big files vs. small files, etc).

Hence, we believe integration of our algorithms would then be beneficial. To this end we see two areas of investigation, which go in parallel with model refinements. First, we plan to focus on a specific scenario in terms of content characteristics. We want to study and characterize scenarios such as software package download in the open source community, or music / video download. We need to identify parameters such as size of the content, rate of obsolescence/ update, request/ access patterns from users, distribution of content across several providers. Second, we will study possible ways to deploy our mechanisms into some existing open source software package, so that they will eventually be implemented and used by real users. The proposed mechanism could be implemented as part of a peer-to-peer cache functionality (such as [15]), or as part of a server cache (such as [16] or [18]). We aim to survey such existing projects in detail to assess the viability of this.

One possible approach would be to take one of the surveyed open source projects, and extend it with our proposed self-organization mechanisms. Working with one of the projects who are already deployed (although in the academic community) on several hundreds of machines in PlanetLab (such as Coral) would be a plus, relieving us from the need to setup a test-bed. We will analyze the trade-off between applicability of our algorithms to the different projects, accessibility of the software, and ease of deployment.

4 Conclusion

We have presented two application domain models and given some initial results from them. We have identified a number of open issues that require further work if more specific implementation orientated models are to be developed and they are to be applied toward implementations. Also we have outlined some directions for generic methods and techniques that might be applied in developing abstract biological and social models towards applications. We are far from a general design method but believe we have outlined some of the broad features that such a future method might embody: chains of models, design patterns and utility grounding methods.

References

[1] Arteconi, S.; Hales, D. (2006) Broadcasting at the Critical Threshold. *Technical Report UBLCS-2006-22, University of Bologna, Dept. of Computer Science.* [DELIS-TR-0373]

[2] Arteconi, S.; Hales, D. (2005) Greedy Cheating Liars and the Fools Who Believe Them. *Technical Report UBLCS-2005-21, University of Bologna, Dept. of Computer Science.* [DELIS-TR-0397]

[3] Babaoglu, O., Canright, G., Deutsch, A., Di Caro, G., Ducatelle, F., Gambardella, L., Ganguly, N., Jelasity, M., Montemanni, R., Montresor, A., and Urnes, T. (2006) Design Patterns from Biology for Distributed Computing. In *ACM Transactions on Autonomous and Adaptive Systems, vol. 1, no. 1, 26–66.*

[4] Callaway, D. S., Newman, M. E. J., Strogatz, S. H., and Watts, D. J. (2000) Network robustness and fragility: Percolation on random graphs. *Physical Review Letters, 85:5468*

[5] Hales, D. and Babaoglu, O. (2006) Towards Automatic Social Bootstrapping of Peer-to-Peer Protocols. *ACM SIGOPS Operating Systems Review (Special Issue on Self-Organizing Systems)* 40(3). [DELIS-TR-0371]

[6] Hales, D. (2006) Emergent Group-Level Selection in a Peer-to-Peer Network. *Complexus 2006;3.: 108-118.* [DELIS-TR-0200]

[7] Hales, D. and Arteconi, S. (2006) SLACER: A Self-Organizing Protocol for Coordination in P2P Networks. *IEEE Intelligent Systems* 21(2):29-35. [DELIS-TR-0368]

[8] Hales, D. and Edmonds, B. (2005) Applying a socially-inspired technique (tags) to improve cooperation in P2P Networks. *IEEE Transactions in Systems, Man and Cybernetics - Part A: Systems and Humans, 35(3):385-395.* [DELIS-TR-0111]

[9] Hales, D. (2000) Cooperation without Space or Memory: Tags, Groups and the Prisoner's Dilemma. In *Moss, S., Davidsson, P. (Eds.) Multi-Agent-Based Simulation. Lecture Notes in Artificial Intelligence 1979.* Berlin: Springer-Verlag.

[10] Jelasity, M., Montresor, A. and Babaoglu, O. (2005) Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst., 23(1):219-252*

[11] Moore C., and Newman, M. (2000) Exact solution of site and bond percolation on small-world networks. *Physical Review, 62:7059-7065.*

[12] Sasson, Y., Cavin, D. and Schiper, A. (2003) Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2003).*

[13] Peersim Peer-to-Peer Simulator, http://peersim.sf.net

[14] The BISON Project, http://www.cs.unibo.it/bison

[15] Iyer, S., Rowstron, A., Druschel, P.: Squirrel (2002) A decentralized, peer-to-peer web cache. In *Proc of the 21st Ann ACM Symp on Principles of Distributed Computing*, ACM.

[16] *www.coralcdn.org*

[17] *www.codeen.cs.princeton.edu*

[18] *www.globule.org*