From Selfish Nodes to Cooperative Networks – Emergent Link-based incentives in Peer-to-Peer Networks[1]

David Hales
University of Bologna, Italy
dave@davidhales.com

Abstract

For Peer-to-Peer (P2P) systems to operate effectively peers need to cooperate for the benefit of the network as a whole. Most existing P2P systems assume cooperation, relying on peers to perform tasks that are of no direct individual benefit. However, when large open systems are deployed such assumptions no longer hold because by adapting selfishly nodes may become "freeloaders" leaching resources from the network. We present initial results from simulations of an algorithm allowing nodes to adapt selfishly yet maintaining high levels of cooperation in both a Prisoners' Dilemma and a flood-fill query scenario. The algorithm does not require centralized or third party reputation systems, the monitoring of neighbor behavior or the explicit programming of incentives and operates in highly dynamic and noisy networks. The algorithm appears to emerge its own incentive structure.

1. Introduction

Open Peer-to-Peer networks (in the form of applications on-top of the internet) have become very popular for file sharing applications (e.g. Kazaa [10], Gnutella [4]). However, as has been shown [1], in such file sharing scenarios we find that a majority of users do not actually share their own files (they act selfishly). However, these networks are still popular because it only requires a minority to share high quality files for all to benefit - a small amount of altruism appears to be enough to support some level of service for file-sharing applications. But is this situation inevitable? Can selfish nodes be discouraged? One solution is to have a closed system in which we can ensure that each node runs a particular peer application, hard-coded to be cooperative. But this option precludes the benefits of open systems. In open systems the protocols are open so any node that understands the protocol can participate. This allows for truly decentralized control and freedom for innovation (new nodes with new kinds of behavior may enter the network). A desirable goal would be to have an open network with each node benefiting from common resources yet still (somehow) discourage free loaders.

Many possible mechanisms exist to solve this problem such as trusted third parties [13], the generation, storage and sharing of reputation information [11] and the application of reciprocal punishments (using the shadow of the future [2]) with on-going interaction partners (in this latter instance recent interest has focused on incentive structures [3, 14]). In each case these solutions require either the services of some external centralized (trusted) authority or various levels of overhead based on tracking, storing and processing the behavior of on-going interactions with other nodes. In the case of incentives, an appropriate incentive system needs to be designed *a priori* and on-going interactions with other peers monitored.

The problem we are addressing here is a "commons tragedy" [7], something that the social and biological sciences have been exploring for some time [8, 18, 17]. In this paper we adapt and apply some recent mechanisms advanced within the social sciences [5, 15, 16] that promote cooperation without the previously mentioned overheads.

The very simple algorithm presented here requires no such overheads but still keeps free loading at a very low level even though nodes act selfishly and are free to adapt their behaviors and move around the network by changing their links to other nodes. We find that our adapted algorithm forms a kind of

[1] This work partially supported by the EU within the 6th Framework Program under contract 001907 (DELIS).

emergent incentive structure – without the associated overheads.

The paper proceeds as follows: in section two we describe the basic algorithm; in section three we apply the algorithm to the single-round Prisoner's Dilemma (PD) game; in section four we simulate a more realistic flood-fill query process in a P2P file sharing scenario – we show how the algorithm suppresses free riding by stopping peers from flooding the network with queries. Finally in section five we summarize related work and in section six we conclude with open issues and future work.

2. Tags and the algorithm

The basic algorithm has been adapted from previous (quite different) simulation work using "tags". This work demonstrates a novel method of maintaining high levels of cooperation in environments composed of selfish, adaptive agents [5, 6, 15, 16]. The emphasis of the previous work has been towards understanding biological and social systems. Tags are markings or social cues that are attached to individuals (agents) and are observable by others [9], often represented in models by a single number, they evolve like any other trait in a given evolutionary model. The key point is that the tags have no direct behavioral implication for the agents that carry them. But through indirect effects, such as the restriction of interaction to those with the same tag value, they can evolve from initially random values into complex ever changing patterns that serve to structure interactions. The simulated environments in which tags have been applied have generally been very simple with interactions based on pair-wise games with immediate payoffs [6, 15]. Never the less, we have attempted to adapt the salient features of such tag systems for application in P2P networks. These features are that agents:

- restrict interact to those with whom they share a group defined by tag value
- selfishly and greedily optimize by preferentially copying the behavior and tag of others with higher utility
- periodically mutate their tags and behaviors

By copying and mutating tags, agents effectively move between interaction groups. By restricting interaction within groups *free riders* tend to kill (reduce the membership) of their own group over time because exploited agents will tend to move elsewhere to get better payoffs, while cooperative groups tend to spread via mutation of the tag. Previous tag models have demonstrated high levels of cooperation in "commons tragedy" scenarios (e.g. in the Prisoners Dilemma – see below). We will not cover the results of the previous tag models in detail here, since the emphasis is not relevant and space precludes detailed treatment, rather we will present our newly derived algorithm (based on the salient features outlined above) and the results we obtained when applying it to two different simulated P2P scenarios.

2.1 SLAC - the basic algorithm

The basic algorithm assumes that peer nodes have the freedom to change behavior (i.e. the way they handle and dispatch requests to and from other nodes) and drop and make links to nodes they know about. In addition, it is assumed nodes have the ability to discover other nodes randomly from the network, compare their performance against other nodes and copy the links and (some of) the behaviors of other nodes.

For the purposes of this paper (as discussed previously) we assume that nodes will tend to use their abilities to selfishly increase their own utility in a greedy and adaptive way (i.e. if changing some behavior or link increases utility then nodes will tend to select it). The algorithm relies on Selfish Link and behavior Adaptation to produce Cooperation (SLAC) - a rough outline is given below:

Over time nodes engage in some activity and generate some measure of utility U (this might be number of files downloaded or jobs processed etc, depending on the domain).

Periodically, each node (i) compares its performance against another node (j), randomly selected from the population. If $Ui < Uj$ node i drops all current links and copies all node j links and adds a link to j itself. Also, periodically, and with low probability, each node adapts its behavior and links in some randomized way using a kind of "mutation" operation. Mutation of the links involves removing all existing links and replacing them with a single link to a node randomly drawn from the network. Mutation of the behavior involves some form of randomized change - the specifics being dictated by the application domain (see later).

Previous tag models, on which SLAC is based [5] have indicated that the rate of mutation applied to the links needs to be significantly higher than that applied to the behavior (by about one order of magnitude).

When applied in a suitably large population, over time, the algorithm follows a kind of evolutionary process in which nodes with high utility tend to replace nodes with low utility (with nodes periodically changing behavior and moving in the network). However, as will be seen, this does not lead to the dominance of selfish behavior - as might be intuitively expected – since a form of incentive mechanism emerges via a kind of ostracism in the network.

3. Dilemmas on the network

Since previous tag models have demonstrated high levels of cooperation in the single-round Prisoner's Dilemma (PD) game we initially test the SLAC algorithm on this domain – in order to determine if our adapted algorithm (for application within networks) still produces the desirable cooperation supporting results. The PD game captures very minimally a "commons tragedy". First we introduce the PD then we describe a simulation applying SLAC to the PD within a network scenario. Here we focus on the single round PD *not* the Iterated PD (IPD). We consider that the single round is a more general form and form applicable to dynamic P2P scenarios (see conclusion). In any case, we do not assume results from the PD are necessarily applicable so we test those results in a more realistic P2P file-sharing scenario based on a previously presented model [14] (see section 4 below).

3.1. The Prisoners Dilemma

The Prisoner's Dilemma (PD) game captures a scenario in which there is a contradiction between collective and self-interest. Two players interact by selecting one of two choices: Either to "cooperate" (C) or "defect" (D). For the four possible outcomes of the game players receive specified payoffs. Both players receive a reward payoff (R) and a punishment payoff (P) for mutual cooperation and mutual defection respectively. However, when individuals select different moves, differential payoffs of temptation (T) and sucker (S) are awarded to the defector and the cooperator respectively. Assuming that neither player can know in advance which move the other will make and wishes the maximize her own payoff, the dilemma is evident in the ranking of payoffs: T > R > P > S and the constraint that 2R > T + S. Although both players would prefer T, only one can attain it. No player wants S. No matter what the other player does, by selecting a D move a player ensures she gets either a better or equal payoff to her partner. In this sense a D move can't be bettered since playing D ensures that the defector cannot be suckered. This is the so-called "Nash" equilibrium for the single round game. It is also an evolutionary stable strategy for a population of randomly paired individuals playing the game where reproduction fitness is based on payoff. So the dilemma is that if both individuals selected a cooperative move they would both be better off but both evolutionary pressure and game theoretical "rationality" selected defection.

3.2. PD on the network

In order to discover if SLAC supports cooperation in the single round PD game we simulated the following scenario. We initialize a random graph of size N (with node degree $k = 20$). Each node stores a single round PD strategy (either to cooperate or defect) initially we set all strategies to defect. Time is structured into cycles. In each cycle each node (i) is select from the network in turn and "fired". When i is fired it randomly selects one of its neighbors (j). If i has no neighbors then a node (j) is selected from the population randomly and wired as a neighbor. Nodes i and j then play a single-round of PD based on their strategies and receive the appropriate payoff (either T, R, P or S).

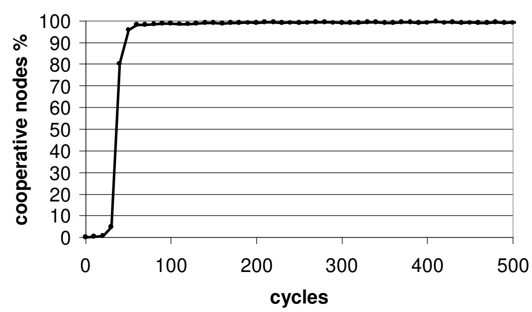

Figure 1: A typical run for a 10^4 node network

After each node has been fired (one time period) the previously described SLAC algorithm is applied in the following manner: N / 2 pairs of nodes (i, j) are selected from the population at random with replacement. If $Ui > Uj$ then node j drops all existing links and copies node i's links additionally linking to node i itself. Also j copies the strategy of i. If $Ui < Uj$ then the mirror process is performed (i copying j). If

$Ui = Uj$ then a randomly selected node (i or j) is designated "winner" and the process proceeds as if that node had a higher U value. The node utility value was set equal to the *average game score* obtained by a node in the time period.

After any node i copies another node j it applies mutation to (i.e. may randomly change) both links and the game strategy with low probability. With probability m, the strategy is changed. With probability $10m$ the links from node i are removed and replaced with a single link to a randomly chosen node from the population.

We set the maximum node degree as 20 links. If any node requires a further link then it simply removes an existing link at random and proceeds. The mutation rate was set to m=0.001. The PD payoffs were set to T = 1.9, R = 1, P = S = d (where d is some small value greater than zero) We use these values for simplicity. When a small value is added to P enforcing the strict inequalities of the PD then no significant differences are observed.

3.3. Results

Figure 1 shows a time series of a single typical run for a network of size $N = 10^4$. The y-axis shows the percentage of nodes in the entire network that are cooperators. Figure 2 compares a set of runs showing the number of cycles (for different network sizes) required to reach 99% cooperative nodes (in all cases we initialize the population as all defectors). For each network size 10 runs are given. As can be seen the algorithm appears to scale without increasing time to high cooperation (further runs to for networks of size $N > 10^5$ showed the same results).

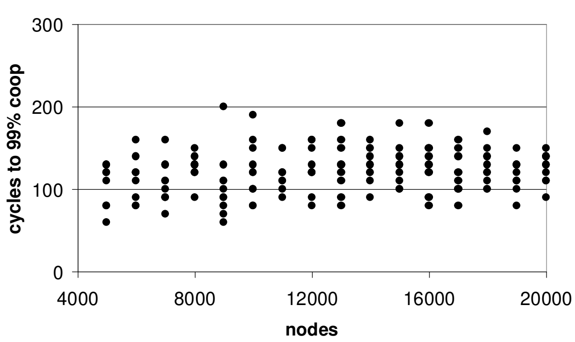

Figure 2: Cycles to high cooperation

It appears that the SLAC algorithm does indeed manifest some of the desirable properties from the tag models on which it was based. In the network simulation given here we have achieved high cooperation in the single-round PD game when played with neighbors *even though nodes are behaving selfishly, trying to maximize their utility* by copying other nodes who get higher utility.

4. A file sharing scenario

In this section we outline the results of applying SLAC to query answering in a simulated P2P file-sharing scenario. The scenario is a simplified form of that given in [14]. It models a flood-fill query process where nodes periodically generate and forward queries to their neighbors. Neighbors either deal with the these queries (by producing a hit or forwarding it to all their neighbors) or ignore the query.

By translating the algorithm into a more realistic P2P file-sharing scenario we test if the previous results obtained from the relatively abstract single round PD game carry-over. We do not assume that results gained in synchronous, abstracted PD type domains will necessarily translate into the more asynchronous and less clearly delineated utility domains that reflect engineering realities.

After [14] each node (i) is defined by three state variables: an answering power Ai, a questioning power Pi and a capacity Ci, where both Ai and Pi are in the range [0..1] and Ci takes some cardinal value greater than zero. Each variable quantifies the behavior of a node over some unit of time t. Ci indicates the capacity of the node in total number of queries (we assume that generating a query or answering a query take one unit of capacity). Pi givens the proportion of the capacity Ci that will be allocated to generating new queries. Conversely, $1 - Pi$ of the capacity will be allocated to answering queries from other nodes. The answering capacity Ai gives a probability that a node can directly match a query (producing a hit). It represents indirectly the amount and quality of files served by the node.

For experiments given here all nodes have fixed values of Ai = 0.4 and Ci = 100 but we allow Pi to be adapted by the node (see below).

Over a single time period each node i may process a total of Ci queries. This capacity is divided between generating $Pi \bullet Ci$ new queries (passed to neighbor nodes) and reserving enough capacity to process $(1-Pi) \bullet Ci$ queries from neighbors. Pi therefore represents a kind of measure of selfishness. If $Pi = 1$ then node i uses all its capacity to generate new queries - ignoring queries from neighbors. If $Pi = 0$

then i uses all its capacity processing queries from neighbors.

4.1. Outline of a simulated time period

In a simulated time period, $C \bullet N$ nodes (where N is the number of nodes in the population) are selected randomly from the population (with replacement) and "fired". If a fired node still has capacity to generate queries it generates one query and passes this to its neighbors otherwise the node takes no action. When a node (i) receives a query, if it has spare capacity, it processes the query. With probability Ai a "hit" is produced for the query. If no hit is produced the query is passed to the neighbor nodes of i. If a node has no capacity left to process a query it is ignored – no action is taken. Queries are not passed on indefinitely but have a preset "time-to-live" (*TTL*) after which they are ignored by all nodes. In all experiments presented here $TTL = 3$ - meaning that queries never get more than a maximum of 3 nodes depth from the originating node. The process of firing nodes in random order with replacement introduces noise in the form of some nodes firing more often than others and some nodes not being able to generate their full quota of queries. We view this as reasonable since it introduces realistic kinds of noise such as non-synchronized nodes with differential processing speeds etc.

4.2. Application of SLAC

After each time period (that is after $N \bullet C$ node firings - see above) the previously described SLAC algorithm is applied. N / 2 pairs of nodes (i, j) are selected from the population at random with replacement. If $Ui > Uj$ then node j drops all existing links and copies node i's links additionally linking to node i itself. Also Pj is set to Pi (copying the query handling behavior of i). If $Ui < Uj$ then the mirror process is performed (i copying j). In the case $Ui = Uj$ then a randomly selected node (i or j) is designated "winner" and the process proceeds as if that node had a higher U value.

For the experiments present here we used a utility value equal to the *total number of hits* obtained by a node in the time period. Obviously this would tend to be higher if P was higher (generating more queries). We used the utility value of total hits (per node) since this gives an apparent incentive for freeloading. If the average hits per query is used as the utility then there is no commons tragedy – because nodes wont generally increase their utility by performing more queries.

After any node i copies another node j it applies mutation, with low probability, to the links and the Pi value. With probability m, Pi is changed to a random value selected uniformly from the range [0..1]. With probability $10m$ the links from i are removed and replaced with a single link to a randomly chosen node from the population (see above).

4.3. Summary of simulation scenario

For the purposes of simulation we represent the network as an undirected graph in which the degree of any node is fixed at a maximum value (20 in all cases). When any operation requires a further link from a node, the node simply deletes a randomly chosen existing link and continues with the new link operation. We experimented with various initial topologies for the graph, including randomly connected, lattice, "small world" and completely disconnected. All produced similar results the ones we present here. We also experimented with different initial P values. Again we found we obtained similar results (even when all P values are initially set to zero – see later). Results given (unless otherwise stated) start with initially random graphs and randomly selected P values.

4.4. Results

In order to gain a benchmark which measures how the network behaves without the application of the SLAC algorithm we ran 10 trials for 10 cycles on static networks with randomly initialized topologies and P values. We did this for a number of network sizes N = [200..51200]. All other values were kept as previously described. Since in the static case nothing changes, the averaging over 10 cycles is done simply to smooth out the stocasticities of the model. Averaging over 10 different trials (with unique pseudo-random number seeds) smoothes out the different initial network topologies and P values.

We considered the following two measures: the average number of queries generated per node in a cycle (nq) and the average number of hits per node generated per cycle (nh). We found that with low variance $nq = 49.45$ and $nh = 20.13$ in all cases. Calculating nh / nq gives an average hit rate per query generated = 0.41. We might expect $nq = 0.5$ since the P values are selected uniformly randomly but this slightly lower value is a result of the (random selection with replacement) method of firing nodes as described earlier.

Given these baseline values for *nq* and *nh* we can investigate the effect of applying the SLAC algorithm. If results give a consistently higher number of hits (*nh*) by keeping the number of queries generated (*nq*) low then SLAC is suppressing the self-interest of the nodes and thus benefiting of the network as a whole.

Figure 3 shows a time series for a typical run (with a network of size $N=10^4$) with SLAC enabled.

As can be seen, over time, *nq* decreases and *nh* increases. Notice also that initially these values move in the opposite direction, indicating an initial favoring of selfish behavior, but this is soon corrected. This shows that the evolutionary process (forming cooperative groups within the network) takes a few cycles to get started from the initially randomly initialized network.

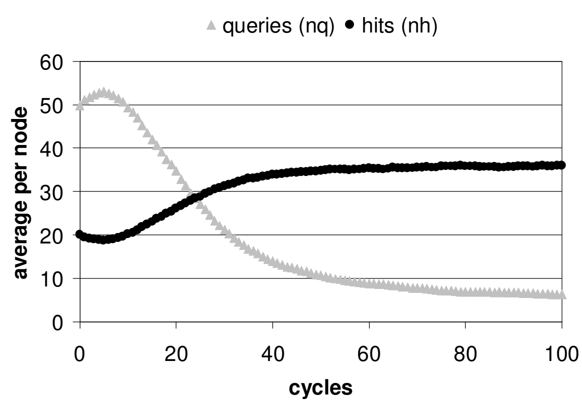

Figure 3: A typical run for a 10^4 node network

Figure 4 shows the same measures averaged over cycles (40-50) for different network sizes (with 10 independent runs for each network size). As can be seen, most runs follow a similar pattern to that shown in figure 3. However, notice that as size of the network increases the variance of the individual runs decreases – indicating that larger networks are less sensitive to on-going stocasticities and initial conditions.

Figure 5 shows the number of cycles before high hit values are attained (when *nh* > 30). Again 10 independent runs are shown for various network sizes. As before the variance of results decreases as network size increases and there is no significant increase in the number of cycles required for larger networks - suggesting that SLAC scales well in this scenario also.

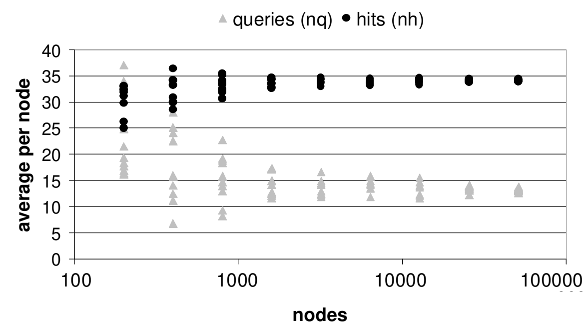

Figure 4: Results (averaged over cycle 40..50) for different network sizes (10 individual runs for each network size)

Overall then, our initial experiments suggest that SLAC *does indeed control the self-interest of the nodes* by keeping down the number of queries generated and hence increasing the total hits overall.

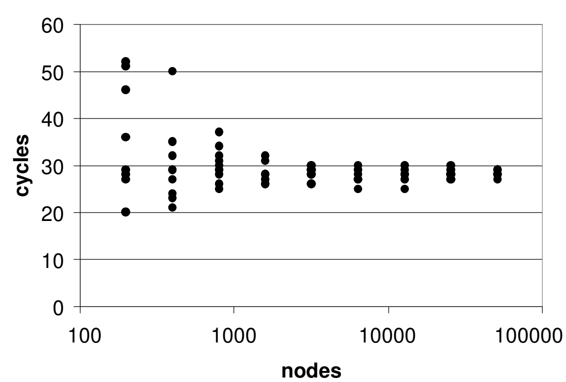

Figure 5: Cycles to high hit values (nh > 30) for different network sizes (10 runs each)

5. Related work

SLAC bears similarities to the more complex "SLIC" algorithm [14]. However, in that work the incentive structures are explicitly programmed, with each node monitoring the service it receives from its neighbors and updating weights which moderate the future service it offers to others. There is therefore explicit retaliation programmed into the model (as applied in [3]). For the model presented in this paper the incentives effectively *emerge* from the dynamic behavior of the nodes (moving in the network) rather than being explicitly programmed in. Nodes therefore not need to monitor or store the performance of others

– reducing overheads. In [14] simulations are only applied to scenarios in which single "probe nodes" behave selfishly – nodes do not adapt their behavior to increase their utilities network wide. Consequently it is not clear how the model would react when *all nodes* are acting selfishly rather than just a small number (however, this is mentioned as future work).

Previous models inspired by game theoretical approaches offer some similar insights [12] but in this work network topology is not explicitly modeled and the strategies relay on repeated game strategies (i.e. tit-for-tat in the Iterated PD such). Such iterated strategies require on-going interactions with recognizable individuals – our model does not rely on this since it is based on mechanisms that work well in the single-round game.

Our model bears comparison to the social simulation of leadership dynamics presented in [19]. However, this model is deterministic and relies on agents knowing the strategies of others when moving. Never-the-less, agents in the model move around a social network, making and breaking links based on self-interest and play the single round PD – insights from this model may be applicable to SLAC (the comparison process is on-going work).

6. Conclusion and future work

It seems that the desirable properties from the previously discussed tag models [5, 6, 15] have been usefully carried over into a dynamic network scenario. The SLAC algorithm potentially offers a very generally applicable mechanism for controlling selfish behavior in many possible P2P task domains – without the need to program and test explicit incentive mechanisms for each domain. In the work presented here *the SLAC algorithm effectively emerges an incentive mechanism from the selfish moving behavior of the nodes* - by ostracizing selfish nodes over time. This happens because although a selfish node may do well for a while it will tend to lose its exploited neighbors as they find other nodes that are members of more cooperative groupings and hence have higher utilities.

These results are more generally applicable than strategies based on the iterated PD (IPD) such as tit-for-tat (TFT) because no past or future interaction with the same nodes is required. This is of particular value in highly dynamic networks where nodes are constantly entering and leaving – hence interacting is often with strangers. Additionally, this saves on the overhead of remembering past interactions or identifying specific nodes. All this adds to flexibility and scalability.

Interestingly if we imagine SLAC being deployed in a more stable network (or sub-network) in which nodes did practice TFT-type strategies then SLAC would perform as well as TFT players – since cooperation would be rewarded with cooperation.

Currently however, SLAC requires some *a priori* utility function that can be evaluated by each node and (perhaps less realistically) compared between nodes. It also requires that nodes can copy the behaviors and links of other nodes. This latter process can be interpreted in two ways, like [12] we can say that this process represents the selfish behavior of users – copying (possibly modified) peer-client software that appear to offer increased performance increases. If we offer this interpretation then the higher mutation applied to the links over the behaviors (recall these were set 10 times higher – this carried over from the tag models) implies that peers can change links more quickly than client software. This certainly makes sense since adapting software is generally time consuming (mutation on behaviors) but changing links may not be (simply resetting peer-clients). An alternative interpretation is that the process is an automatic peer level process. In this case the assumptions of node copying may be unrealistic and raise a number of security issues.

We have not tested SLAC against the introduction of "whitewashers" [12] - nodes that never change their behavior from selfish options. We have assumed that nodes will act greedy and adopt behaviors that are producing higher utility for other nodes – even if those behaviors are non-selfish. Future work will test the robust of the model against various proportions of whitewashers – we speculate that some proportion of the population will be tolerated but that a large number may significantly damage the formation of cooperation in the system.

In the worse case it would seem that "smart" whitewashers that constantly probe for higher scoring neighborhoods, exploiting and then moving on could be the worse enemies of a SLAC like approach. In order to deal with such problems it is necessary to place clear limits and interpretations on when and how nodes get access to other nodes for copying. Again this is on-going work.

Here we have presented only some initial experiments with simulations of SLAC. Although these look encouraging more investigation is required. For example, in the previous tag algorithms on which SLAC was based – it was possible to visualize the formation and dissolution of cooperative groups over time and even to characterize analytically, some of the properties (such as time to cooperation). However, it is unclear how to do this on a graph topology, where groupings overlap. A start along this line would be to begin to characterize the topological evolution of the network. Some of our very initial results in this direction suggest that the network tends to become disconnected into a number of components – this might have implications for the applicability of SLAC in many domains – but this is under investigation.

The experiments presented here suggest that the SLAC algorithm may offer potential for reducing freeloading in P2P networks and / or understand why such freeloading is not already dominating some existing networks. In either case we believe that our initial results are promising. Our continuing work will address the open issues, specifically to make SLAC more realistically applicable to actual task domains.

References

[1] Eytan Adar and Bernardo A. Huberman Free Riding on Gnutella. *First Monday* Volume 5, No. 10. 2000.

[2] R. Axelrod The Evolution of Cooperation, Basic Books, New York. 1984.

[3] B. Cohen. Incentives build robustness in bittorrent. In Workshop on Economics in Peer-to-Peer Systems, 2003.

[4] Gnutella. http://www.gnutella.com/

[5] D. Hales. Cooperation without Space or Memory: Tags, Groups and the Prisoner's Dilemma. In *Moss & Davidsson, (eds.) Multi-Agent-Based Simulation. LNAI 1979:157-166*. Springer. Berlin. 2000.

[6] D. Hales. & B. Edmonds. Evolving Social Rationality for MAS using "Tags", In *Rosenschein et al. (eds.) Proceedings of the 2nd International Conference on Autonomous Agents and Multi-agent Systems, (AAMAS03)*, ACM Press, 497-503. 2003.

[7] Garrett Hardin "The Tragedy of the Commons,", *Science*, 162:1243-1248. 1968.

[8] Thomas Hobbes, *Leviathan*, ed. by J.C.A. Gaskin, Oxford, 1998.

[9] J. Holland. The Effect of Labels (Tags) on Social Interactions. Santa Fe Institute Working Paper 93-10-064. Santa Fe, NM, 1993

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

[11] S. D. Kamvar, M. T. Scholosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In the 12th International WWW Conference, 2003.

[12] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In Workshop on Economics of Peer-to-Peer Systems, 2003.

[13] MojoNation. http://www.mojonation.com

[14] Qixiang Sun & H. Garcia-Molina SLIC: A Selfish Link-based Incentive Mechanism for Unstructured Peer-to-Peer Networks. In *Proceedings of the 24th IEEE international Conference on Distributed Systems*. IEEE computer Society. 2004.

[15] R. L. Riolo, M. D. Cohen. & R. Axelrod. Evolution of cooperation without reciprocity. *Nature* 414, 441-443. 2001

[16] K. Sigmund. & M. A. Nowak. Tides of Tolerance. *Nature* 414, 403-405. 2001.

[17] J. Maynard Smith. Evolution and the Theory of Games. Cambridge University Press. Cambridge. 1982.

[18] R. Trivers. The evolution of reciprocal altruism. *Q. Rev. Biol.* 46, 35-57. 1971

[19] M. G. Zimmermann, Victor M. Egufluz and Maxi San Miguel. Cooperation, adaptation and the emergence of leadership. In *`Economics with Heterogeneous Interacting Agents', pp. 73-86, A. Kirman and J.B. Zimmermann (eds.)*, Springer, Berlin. 2001.