Greedy Cheating Liars and the Fools Who Believe Them[⋆]

Stefano Arteconi, David Hales, and Ozalp Babaoglu

University of Bologna
Dept. of Computer Science
{arteconi,hales,babaoglu}@cs.unibo.it

Abstract. Evolutionary algorithms based on "tags" can be adapted to induce cooperation in selfish environments such as peer-to-peer systems. In this approach, nodes periodically compare their utilities with random other peers and copy their behavior and links if they appear to have better utilities. Although such algorithms have been shown to posses many of the attractive emergent properties of previous tag models, they rely on the honest reporting of node utilities, behaviors and neighbors. But what if nodes do not follow the specified protocol and attempt to subvert it for their own selfish ends? We examine the robustness of a simple algorithm under two types of cheating behavior: a) when a node can lie and cheat in order to maximize its own utility and b) when a node acts nihilistically in an attempt to destroy cooperation in the network. For a test case representing an abstract cooperative application, we observe that in the first case, a certain percentage of such "greedy cheating liars" can actually improve certain performance measures, and in the second case, the network can maintain reasonable levels of cooperation even in the presence of a limited number of nihilist nodes.

1 Introduction

Recent evolutionary "tag" models developed within social simulation and computational sociology [7,16] have been proposed as possible candidates for robust distributed computing applications such as peer-to-peer file sharing [9]. These models exhibit a mechanism by which even greedy, local optimizers may come to act in an unselfish manner through a "group like" selection process.

Evolutionary models rely on the concept of a "fitness" or "utility" which can be derived by each individual within a population, usually based on how well it is performing some task. Variants of behavior within the population are assumed to be selected and reproduced in proportion to relative utilities, resulting in fitter variants growing in the population.

In order to apply these evolutionary models to distributed systems, we need to capture the notion of fitness or utility, as well as the process of differential

[⋆] Partially supported by the EU within the Sixth Framework Programme under contract 001907 "Dynamically Evolving, Large Scale Information Systems (DELIS)".

spreading of behavioral variants. In previously proposed P2P applications, each node derived its utility from some measure of on-going performance. For example, in a file-sharing application, the number of successful downloads can be used [9]. Nodes then periodically compared their utility with another peer chosen randomly from the population. Of the pair, the node with the lower utility then copied the behavior and links of the "fitter" peer. This sort of "tournament selection" rule can be seen as "reproducing" fitter nodes and replacing the less fit ones. Alternatively it can be viewed as less fit nodes copying fitter nodes to improve their performance.

For this mechanism to work, however, nodes must behave correctly and report their local state to other nodes honestly. Yet, a node may deviate from its expected behavior or lie to other nodes about any one of the following: its utility, its behavior and its neighbor links (when the other node wishes to copy them). What happens if such "cheating liar" nodes are allowed to enter the network? What if they cheat for their own benefit or act maliciously to destroy cooperation that may exist among other nodes?

In this paper we make a distinction between different kinds of node behaviors. Nodes that follow a protocol specification but act selfishly to increase their own performance rather than cooperating to improve global system performance (e.g. leechers in a file-sharing system) are called *selfish* nodes. On the other hand nodes that do not follow the protocol specification we term *deviant*, while nodes which use their knowledge of the protocol to act against it and to achieve some local goal we call *cheaters*.

In an earlier work, we have shown that an evolutionary inspired protocol called SLACER can achieve and maintain small-world networks with chains of cooperating nodes linking all node pairs (so-called artificial social networks) in a robust manner [8]. In this paper, we study SLACER when it is subjected to four different types of deviant node behavior. In each case, deviant nodes use their knowledge of the SLACER protocol and the application task at hand (the canonical Prisoners' Dilemma game) in an attempt to exploit those that are faithfully following the protocol. The four variants of deviant behavior fall into two classes based on their goals: 1) Greedy Cheating Liars (GCL) try to maximize their own utility; 2) Nihilists (NIH) try to degrade the utility of the entire network, even at their own expense.

We find that the SLACER protocol is surprisingly robust, exhibiting graceful degradation of performance measures even when large proportions of nodes are Greedy Cheating Liars. Even more interestingly, we observe that under certain conditions, GCL nodes actually improve certain network performance parameters. In a way, we can view GCL nodes as performing a sort of "service" for which the honest nodes are "taxed". This observation suggests that P2P applications can perhaps be designed to accommodate certain deviant behaviors rather than trying to detect and isolate them.

Nihilist nodes, on the other hand, are more disruptive for the SLACER protocol and even relatively limited proportions of such nodes can degrade the performance of other nodes significantly.

The rest of the paper is organized as follows: in Section 2 we briefly discuss related work on deviant and selfish nodes. In Section 3 the SLACER protocol is described and its vulnerabilities are presented. In Section 4 experimental results for SLACER with honest nodes are analyzed. We introduce the four deviant behavior variants in Section 5 and in Section 6 present experimental results in their presence. Finally in Section 7 conclusions are drawn.

2 Related Work

Given the decentralized nature of P2P systems and the consequent lack of central control, cooperation between nodes is fundamental. To achieve this goal, different incentives and trust mechanisms have been developed [11,14].

Classical approaches to detect deviant nodes and avoid system inconsistencies include data replication, quorum techniques [12] and distributed state machines (DSM) [4]. Although originally designed for client-server systems to solve problems such as data consistency between replicated servers, such methods have recently been used to address cooperation as well as reliability in closed P2P systems requiring some kind of central authority to manage node identities and capabilities to join the system [1]. On the other hand, open P2P networks, where nodes can freely join the system without any cost, can be extremely large and dynamic due to continuous joining and leaving of nodes and rewiring of their neighbor links. In such environments, the proposed techniques appear not to be feasible. Alternative solutions are given by fully decentralized, probabilistic punishment schemes as in [5], but again dynamicity remains a problem in that a deviant node can subvert such a system by frequently rewiring to different nodes.

A more sophisticated approach, able to deal with free-riding nodes as well as cheating and deviant ones, is described in [6]. Here a distributed shared history mechanism is used to calculate trust values for all nodes including strangers (new nodes). This method can limit the incentives for selfish and cheating behavior as well as deterring colluders (cheating nodes jointly operating to exploit other nodes in the network). This approach is limited, however, because no distinction is made between new nodes joining the system (strangers) because a single entry in the shared history is used to take all of them into account as if they were a single node. This means whitewashers - nodes that constantly adopt new identities to escape detection of their deviant behavior - can potentially make the system expensive to join, in the sense that strangers (even non-deviant ones) may be initially punished as if they were deviant nodes, hence they might be discouraged from joining the network.

Our approach called SLACER (see next section) is based on simple, local node interactions with no need for notions and mechanisms such as trust, incentives and shared histories.

Algorithms based on reciprocity (e.g., the well-known tit-for-tat strategy [3]) require that each node store the last interaction it had with each other node it encounters. The strategy works by punishing non cooperative behaviour in future interactions. This can be a problem in a large and dynamic system where nodes often interact with strangers they have never met before. In this case, tit-for-tat dictates that strangers be treated with unconditional cooperation, and this allows selfish nodes to exploit tit-for-tat. Our solution draws on an evolutionary approach that does not require reciprocity [18,16]. The novel approach used in SLACER obviates the need for maintaining histories of past node behaviour or on-going interactions between the same nodes.

3 The SLACER Protocol

SLACER (Selfish Link Adaptation for Cooperation Excluding Rewiring) is a decentralized, local protocol that builds small-world like artificial social (overlay) networks with high levels of cooperation between nodes [8].

Each node in SLACER has a *strategy* which is an application-defined behavior (for example, how willing the node shares files with others), a *view* which is a bounded-size list of immediate neighbors in the network and a *utility* which is again defined by the application and measures how well the node is performing. SLACER assumes that nodes are able to modify their local views and strategies, as well as copy them from other peers, in an effort to increase their utility. It requires a random peer sampling service `selectPeer()` to be used in reproduction and view mutation steps[1]. The basic steps of the SLACER protocol consist of each node periodically selecting a random peer, comparing the local utility with that of the random peer, and copying the peer's strategy and view if it has a higher utility.

By copying another node's view, a node performs a *rewiring* of its links to a new neighborhood and effectively "moves" to a new location in the network[2]. A node's life is divided into *cycles*. In addition to performing application-specific tasks, at each cycle every node tries to increase its performance by copying nodes with higher utility with a given probability ρ. This action, also called *reproduction* after the evolutionary inspiration, is illustrated in Figure 1.

```
j ← peerSelect() //pick a random node
if i.utility ≤ j.utility then
  for each element e of i.view do
    remove e with probability $\omega$
  i.strategy ← j.strategy
  i.view ← i.view ∪ j.view ∪ j //add j and j.view
  i.utility ← 0       //reset local utility
```

Fig. 1. Slacer reproduction phase pseudocode

After the reproduction phase nodes "mutate". Mutation can affect both node strategy and view. With some (low) probability μ, a node replaces its strategy with a different one. With some other (low) probability ν, a node changes

[1] In our implementation, we use the NEWSCAST protocol for peer sampling[10].

[2] If node view exceeds size-limit random elements are dropped.

its view by removing each link with probability ω (the same value used in the reproduction phase) and then linking to a node selected randomly over the population.

SLACER can be seen as a middleware layer (see Figure 2) for building dynamic artificial social (overlay) networks that are conducive to cooperation. To use SLACER, an application must define an appropriate *utility function* capturing the effective local benefit derived from the behavior adopted by a node (for example, number of packets received in a routing scenario, downloaded files in a file sharing application, detected spam messages in a collaborative spam filter, etc.).

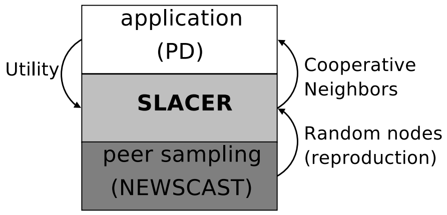


Fig. 2. SLACER middleware architecture

In this paper the Prisoners' Dilemma game is used as a benchmark application to test SLACER performance. This is an interesting and relevant application for our study since it captures the tension between "individual rationality" and "common good" that is typical of many P2P scenarios.

3.1 Prisoners' Dilemma Test Application

The single-round *Prisoners' Dilemma* (PD) game consists of two players selecting independently one out of two possible choices: to *cooperate* (C) or to *defect* (D), and receiving different payoffs according to the four possible outcomes as illustrated in Table 1.

	C	D
C	R, R	S, T
D	T, S	P, P

Table 1. Prisoners' Dilemma payoff matrix

The game captures scenarios where collective interest contradicts the individual one. This game has been selected as a test application because it captures a wide range of possible application tasks where nodes need to establish cooperation and trust with their neighbors without any central authority or coordination. Some practical examples include file or resource sharing, routing messages to facilitate communication between senders and receivers, or warning friends about a virus program and supplying them with a locally available fix.

The dilemma originates from the following constraints among the payoff values under the assumption that players wish to maximize their own payoff: $T > R > P > S$ and $2R > T + S$. Hence although both players would prefer the highest payoff T, only one can obtain it in a single round of the game. When both players select D this leads to the *Nash equilibrium* [13] as well as an *evolutionary stable strategy* (ESS) [17]. Hence, in a population of randomly paired players, a rational selfish player (one that is trying to maximize its own utility) would always play D. If both players select C, they would both perform better (earning R each) than if they both selected D (earning P each) but evolutionary pressure and individual rationality result in mutual defection, so players are pushed to play D and consequently earn P.

In the context of SLACER, the PD test application consists of nodes periodically playing a single-round PD with a randomly selected neighbor from the social network that is constructed by SLACER[3]. A node can only choose between the two pure strategies "always cooperate" or "always defect". The node utility value is obtained by averaging the payoffs received during past game interactions as defined by the PD application. The SLACER protocol then adapts the links and strategies of nodes in an evolutionary fashion as discussed previously.

4 Experimental Results Without Cheaters

In order to obtain a base-line performance, we simulated the SLACER protocol without deviant or cheating nodes — all the nodes follow the protocol faithfully. All of the simulations were performed using *PeerSim*, a P2P overlay network simulator developed at the University of Bologna [15]. We examine cooperation formation (how long it takes for cooperation to spread and the levels reached) and network topology (connectivity, clustering coefficient, etc).

The SLACER parameters used were: $\omega = 0.9, \rho = 0.2, \mu = 0.01$ and $\nu = 0.005$ with a view size limit of 20. Our results show averages over 10 different runs along with 90% confidence intervals when present.

4.1 Cooperation Formation

Figure 3 shows the evolution of cooperation level during a single, typical SLACER run, starting from a random network of all defector nodes. Cooperation reaches very high levels (about 95% of the nodes are cooperating) within a few hundred cycles.

When starting from complete defection, for cooperation to start increasing, it is necessary that two neighboring nodes mutate to cooperating strategies and play a PD round obtaining payoff R. From this point onwards, since cooperating node clusters perform better than defecting node clusters, cooperation rapidly spreads throughout the network by reproduction.

[3] Note that the reproduction phase utilizes a *different* random overlay network: that constructed by the NEWSCAST protocol for peer selection.

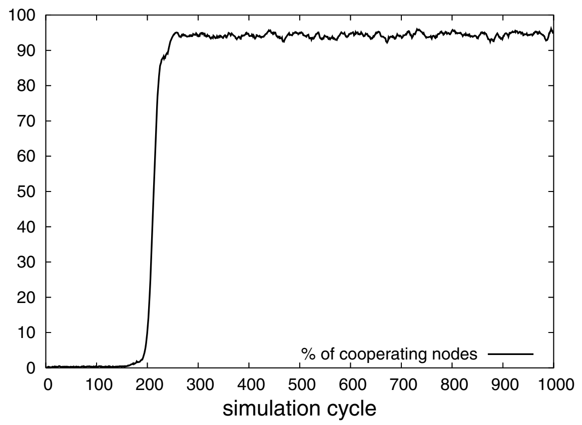


Fig. 3. Cooperation evolution for 4000 nodes

In Figure 4, the time needed to achieve a 95% cooperation level for different network sizes is shown. It is interesting to note that the larger the network, the less time it takes to achieve the same level of cooperation. Furthermore, for larger networks, the results have smaller variance. This is a consequence of fact that the mutation rate and the maximum view size are independent of the network size, and the probability for triggering cooperation as described above increases with network size.

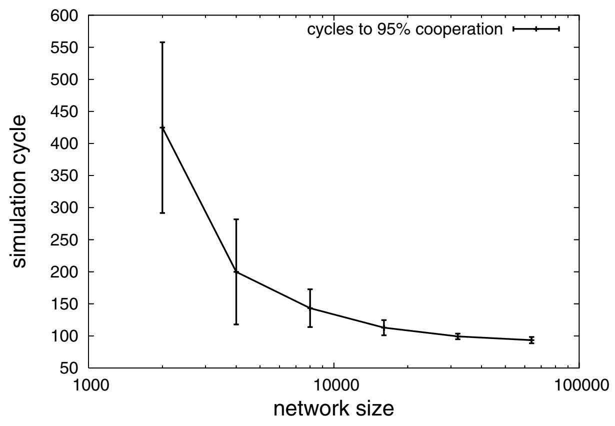


Fig. 4. Time to achieve cooperation for different network sizes

4.2 Network Topology

To analyze properties of the network resulting from SLACER, we adopt traditional topology metrics such as *clustering coefficient* (CC) and *average path length* (APL) in addition to specific metrics *largest cooperative component*

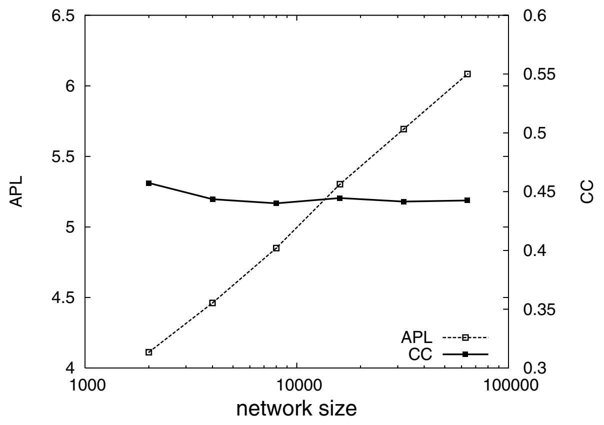


Fig. 5. Clustering Coefficient and Average Path Length for different network sizes

(LCC), *connected cooperative path* (CCP) and *connected cooperative path length* (CCPL).

LCC measures the size of the largest cluster in the subnetwork formed by cooperative nodes only, relative to the total network size. CCP is the proportion of node pairs that are connected through cooperative paths — paths of either length one or that contain cooperating nodes exclusively. CCPL measures the average length of such cooperative paths.

Observing the CC and APL for networks of different sizes produced by SLACER (see Figure 5), we note that they belong to the small world family — CC is high and APL small. Moreover, APL scales log-linearly with respect to network size while CC remains constant suggesting that the network is composed of a growing number of interconnected clusters.

The very large LCC values displayed in Figure 6 indicates a single connected component that accounts for almost the entire network. In other words, partitioning of the network appears not to occur even though SLACER is highly dynamic due to the rewiring of nodes.

Finally, because the CCP values are high and CCPL values are similar to APL (as discussed in [2]), we conclude that selfish nodes in the network (those that have opted to play D in the PD) do not occupy important positions blocking paths between cooperating nodes or resulting in cooperative clusters to be partitioned.

5 Four Kinds of Cheating Nodes

So far we have shown how SLACER can effectively handle selfish nodes — nodes that adopt the defect strategy. What happens if nodes not only act selfishly, but deviate deliberately from the protocol specification?

One way for a node to deviate is to lie about its state to other nodes. When asked during the reproduction phase, a node can report arbitrary values to a peer

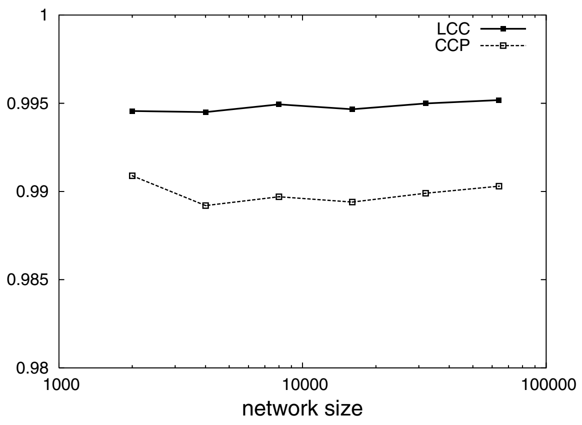


Fig. 6. Largest Connected Component and Cooperative Connected Path for different network sizes

for its strategy, its utility or its view. A deviant node may have the objective to *exploit* or to *destroy* the network. A node that wants to exploit the network wants to maximize its own utility without any regard for the rest of the system. A node aiming to destroy the network wants to lower the utility of all others. We term both such kinds of nodes as *cheating* nodes. In the next sections we outline four possible types of cheating behaviors.

5.1 Greedy Cheating Liars (GCL)

Let us first consider nodes that want to maximize their own utility at the expense of others. In the PD application, but the reasoning applies to any other similar scenario, the maximum possible payoff is obtained by a defecting node when it plays against a cooperating one.

To achieve a high utility then, a node will play the pure strategy "always defect" and never change it. To force its neighbors to behave altruistically, it will report a cooperating strategy when asked. So as not to be excluded from cooperative clusters (through rewiring), it has to rig the utility comparison to guarantee being the "winner" of any utility comparison by another node. This can be easily achieved by lying about utility, reporting always a very large value. Finally, when a node finds itself in a neighborhood that is unsatisfactory (i.e., playing against defectors resulting in an average payoff less than T) it rewires its view causing it to move to a new location in the network.

In summary, a node that wants to exploit the network behaves as follows:

- Always plays D.
- Always declares to be playing C.
- Always declares a high utility (e.g., $2T$).
- Rewires when surrounded by mostly defectors (average payoff less than T).

We call such nodes *Greedy Cheating Liars* (GCL).

5.2 Nihilists (NIH)

Nihilist nodes try to destroy the network by forcing it towards defection, irrespective of their own utilities. To achieve this, a nihilist node itself needs to play the pure strategy "always defect" and bring other nodes to defection as well through a technique similar to GCL nodes.

Since the goal is to spread defection, a nihilist node does not need to lie about it's strategy. On the contrary, it always defects and always reports a defecting strategy. On the other hand, to spread defection and build clusters of only defecting nodes, it needs to win all of the utility comparisons it participates it. This can be obtained by declaring very high utility.

When a nihilist node is surrounded by mostly defectors (i.e., average payoff close to P) it rewires its view trying to destroy other parts of the network.

In summary, a node that wants to destroy the network behaves as follows:

- Always plays D.
- Always declares to be playing D.
- Always declares a high utility (e.g., $2T$).
- Rewires when surrounded by mostly defectors (average payoff close to P).

We call such nodes *Nihilists* (NIH).

5.3 Lying About Views

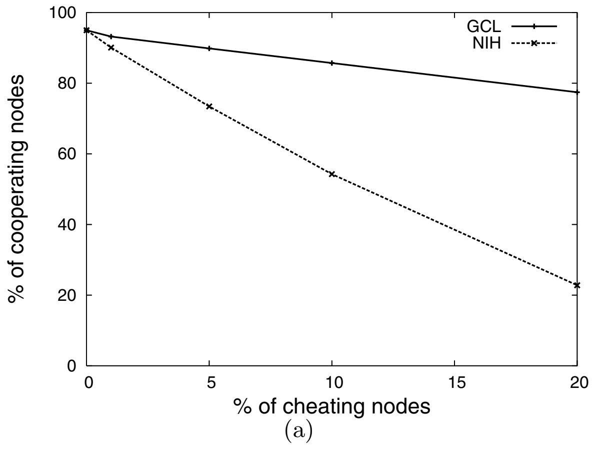
We have also tested what we call "GCL+" and "NIH+" nodes that lie about views as well. When asked for their view, GCL+ nodes report only a link to themselves in an effort to become "hubs" for other nodes and exploit them. NIH+ nodes, on the other hand, report a set of random links from the population as their view in an attempt to spread defection to a wider population. These two cheating variants do not lead to significant differences in cooperation formation behavior with respect to the original GCL and NIH cases. They do, however, affect network topology which we discuss elsewhere [2].

6 Experimental Results with Cheaters

SLACER performance when cheating nodes are present will be now evaluated. We will analyze how cheating nodes influence previously studied cooperation formation and network topology. Moreover, the relative performances of cheating and non-cheating nodes will be compared and discussed.

6.1 Cooperation Formation with Cheaters

Cheating nodes force other nodes to adopt strategies by lying about their state, and as a consequence, cooperation formation follows a different pattern from the case where all nodes are truthful. In Figure 7 the cooperation levels that are achieved and time necessary to achieve them are shown as a function of percentage of cheating nodes in a network of 4000 nodes.



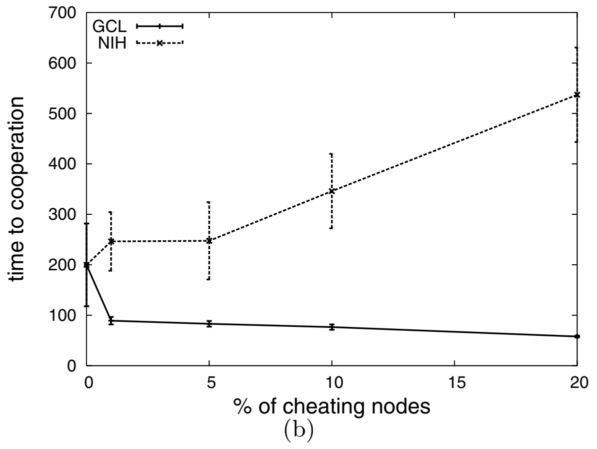


Fig. 7. Cooperation formation with cheating nodes. (a) Cooperation level achieved, (b) Time to achieve cooperation.

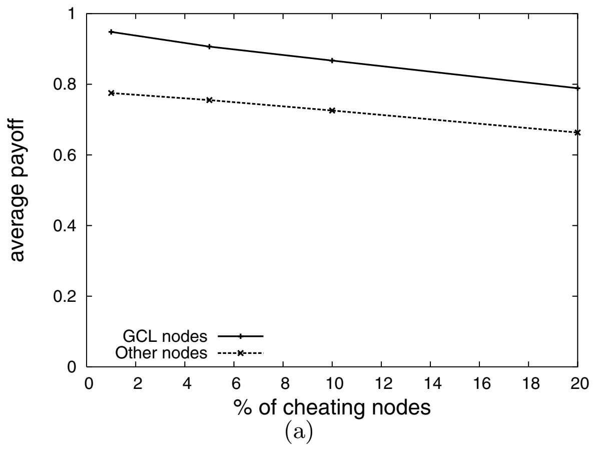
GCL nodes try to force other nodes to be cooperators so that they can exploit them (by defecting). As a result their presence in the network accelerates the spread of cooperation. Almost all the non-GCL nodes become cooperative with high cooperation reached in a shorter time than in a network with no cheaters (see Figure 7(b)).

Having more NIH nodes, on the other hand, increases the time to cooperation. Even though the presence of NIH nodes generally has a negative effect on the network, large numbers of them are needed to bring cooperation down to critical levels as can be seen in Figure 7(a). This indicates the network is resilient though not immune from this kind of attack.

The irregularity in Figure 7(b) for low quantities of NIH nodes results from the fact that NIH nodes make time to cooperation quite unstable (highly variable over different runs) as illustrated by the large confidence interval.

6.2 Network Topology with Cheaters

The presence of GCL and NIH nodes does not affect SLACER network topology formation. Even though cooperation performance could be significantly changed, GCL and NIH nodes use SLACER defined drop and rewiring rules, hence no significant topology modification is involved.



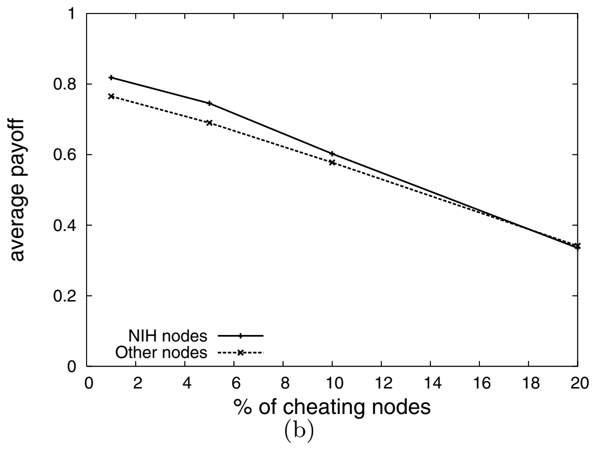


Fig. 8. Utility values in the presence of (a) Greedy Cheating Liars, and (b) Nihilists

6.3 Utilities

Here we consider how node utilities change when there are cheating nodes in the network.

PD payoffs used are $T = 1, R = 0.8, P = 0.1, S = 0$ so average payoff should be equal to 0.8 for a totally cooperating network and 0.1 for a totally defecting one, while the main goal is to have average payoff equal to 1 for GCL nodes, and 0.1 for NIH nodes[4].

Looking at Figure 8, with a small amount of GCL nodes, both cheating and non-cheating nodes have average payoff close to the maximum possible value (1 and 0.8 respectively), but increasing the number of cheating nodes results in both their payoffs decreasing. Normal nodes' payoff decreases because a larger quantity of them is being suckered by cheaters, while GCL nodes' payoff decreases because the more of them there are in the network the more likely two of them end up linking to each other and obtaining payoff $P = 0.1$.

Even for big quantities of GCL nodes there is no dramatic drop in achieved utilities, in fact increasing the number of GCL nodes seems to be more harmful to cheating nodes than to non-cheating ones.

With NIH nodes the average utility decreases linearly with increasing quantity of cheaters (as in GCL), but a large proportion of them is needed to seriously decrease utility (i.e. 20% cheating nodes needed to halve global average payoff value).

7 Conclusions

A protocol called SLACER has been presented to build cooperative artificial social networks in P2P systems. It is based on periodic state comparisons between random nodes and copying of the better-performing node's characteristics.

SLACER has been shown to lead to cooperation, even when initiated in a completely non-cooperative network, and has been tested against four types of malicious, deviant nodes that attempt to subvert the protocol. Such nodes lie about their state according to the goal they want to achieve. Two principle cheating categories have been described: Greedy Cheating Liar (GCL) nodes, whose aim was to optimize their own utility, and Nihilist (NIH) nodes, whose aim was to destroy cooperation in the network.

SLACER was shown to be robust against such behaviors in simulations. Although NIH nodes managed to decrease system performance, a large proportion of them were needed to bring performance under a critical level. Since P2P networks can be very large (thousands or millions of nodes), a relatively large fraction, such as 10%, of cheating nodes would seem improbable — although, of course, certain types of coordinated attacks are possible.

Interestingly, GCL nodes that exploit the network for their own benefit, offered a kind of "service" to SLACER. When GCL nodes were present in the network, cooperation spread faster at the expense of a small decrease in the non-cheating nodes' average payoff.

Robustness against misbehaving entities, as well as cooperation between nodes, are important features for P2P systems since they usually lack any central control and have an open structure. SLACER not only succeeded in reaching

[4] These values are chosen so the PD payoffs could be parameterized over the R value while keeping T, P and S fixed. Results similar to the one presented here were obtained with different R values as long as PD constraints hold ($0.5 < R < 1$).

both of these goals but, from a cooperation formation point of view, it even benefited (for the "time to cooperation" measure) from the presence of cheating nodes trying to exploit the system.

It is important, however, to emphasize that our results show that cheating nodes *do* obtain higher relative utilities than non-cheating nodes. But we assume that cheating nodes would not report honestly their cheating strategy to other nodes since they have a negative incentive to do this. Why would a liar be honest about its lying, and moreover, what would this mean? These are subtle yet crucial points which we expand on in another work [8].

Our results suggests a provocative idea: in open systems, perhaps the best strategy for dealing with malicious cheating nodes is not trying to detect and stop them, but to let them act freely yet turn their misbehaving into a social benefit for the whole system while trying to minimize their damage. Perhaps those who believe what greedy cheating liars tell them are not such fools after all!

Acknowledgements

We are grateful to Mark Jelasity, Simon Patarin, Geoffrey Canright, Alberto Montresor and Gian Paolo Jesi for stimulating discussions, advice and comments.

References

1. A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. Bar fault tolerance for cooperative services. In *Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, October 2005.
2. S. Arteconi and D. Hales. Greedy cheating liars and the fools who believe them. Technical Report UBLCS-2005-21, University of Bologna, Dept. of Computer Science, Bologna, Italy, Dec. 2005. Also available at: `http://www.cs.unibo.it/pub/TR/UBLCS/2005/2005-21.pdf`.
3. R. Axelrod. *The Evolution of Cooperation.* Basic Books, New York, 1984.
4. M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proc. Third Symp. on Operating Systems Design and Implementation*, New Orleans, Feb. 1999.
5. L. P. Cox and B. D. Noble. Samsara: honor among thieves in peer-to-peer storage. In *Proceedings of the ACM Symposium on Operating Systems Principles*, volume 37, 5 of *Operating Systems Review*, pages 120–132, New York, Oct. 2003. ACM Press.
6. M. Feldman, K. Lai, I. Stoica, and J. Chuang. Robust incentive techniques for peer-to-peer networks. In *EC '04: Proceedings of the 5th ACM conference on Electronic commerce*, pages 102–111, New York, NY, USA, 2004. ACM Press.
7. D. Hales. Cooperation without memory or space: Tags, groups and the prisoner's dilemma. In *MABS '00: Proceedings of the Second International Workshop on Multi-Agent-Based Simulation-Revised and Additional Papers*, pages 157–166, London, UK, 2001. Springer-Verlag.
8. D. Hales and S. Arteconi. Slacer: A self-organizing protocol for coordination in peer-to-peer networks. *IEEE Intelligent Systems*, 21(2):29–35, Mar/Apr 2006.

9. D. Hales and B. Edmonds. Applying a socially-inspired technique (tags) to improve cooperation in p2p networks. 35(3):385–395, 2005.
10. M. Jelasity, R. Guerraoui, A.-M. Kermarrec, and M. van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In H.-A. Jacobsen, editor, *Middleware 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 79–98. Springer-Verlag, 2004.
11. S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *12th International World Wide Web Conference*, Budapest, Hungary, 20-24 May 2003.
12. D. Malkhi and M. Reiter. Byzantine Quorum Systems. *The Journal of Distributed Computing*, 11(4):203–213, October 1998.
13. J. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950.
14. T. Ngan, A. Nandi, A. Singh, D. Wallach, and P. Druschel. On designing incentives-compatible peer-to-peer systems. In *T.-W. J. Ngan, A. Nandi, A. Singh, D. S. Wallach, and P. Druschel. On designing incentives-compatible peer-to-peer systems. In Proc. FuDiCo'04*, 2004.
15. PeerSim. http://peersim.sourceforge.net/.
16. R. Riolo, M. D. Cohen, and R. Axelrod. Cooperation without reciprocity. *Nature*, 414:441–443, 2001.
17. J. M. Smith. *Evolution and the Theory of Games.* Cambridge University Press, 1982.
18. R. Trivers. The evolution of reciprocal altruism. *Q. Rev. Biol.*, 46:35–57, 1971.