Study of the Rewiring Factor in an Unstructured P2P

Aminata Bagre[1(✉)], Moustapha Bikienga[2], and Telesphore Tiendrebeogo[3]

[1] University Joseph KI-ZERBO, Ouagadougou, Burkina Faso
aminbagre@gmail.com
[2] University Norbert ZONGO, Koudougou, Burkina Faso
bmoustaph@yahoo.fr
[3] University NAZI BONI, Bobo-Dioulasso, Burkina Faso

Abstract. Social networking sites allow individuals to communicate and share information. Most of P2P systems assume that all peers cooperate for the benefit of the entire network. However, in practice, there is a significant portion of peers that extract resources from the system without contributing in return. Peer-to-peer applications can benefit from human friendship networks (such as e-mail contacts or instant messaging friend lists). However, these networks are not always available. To this end, we study a protocol, called SLACER (Selfish Link-based Adaptation for Cooperation Excluding Rewiring), that allows peer nodes to create their own friendship networks, through random interactions, producing an artificial social network (ASN) where nodes share high trust with their neighbors. To do so, this paper specifically study P2P networks, which constitute the context for the application of the SLACER protocol; an implementation based on the peersim simulator of the rewiring factor W which represents the possibility of rewiring nodes in a peer network; then our main contribution which is the analysis of simulation results obtained on a larger network size with several values of W; finally we do a performance study by comparative study with SLAC (previous version of SLACER).

Keywords: Cooperation · P2P · Social Networks · Self-Organization · Rewiring

1 Introduction

As long as people have existed, whether in their personal or professional lives, they have come together by interest to form networks. Nowadays, these individuals have the possibility to gather online via the Internet, especially on social networks. In [1] the authors state that peer networks open new perspectives on the way we apprehend distributed computing. Indeed, the entities composing

Supported by organization x.

A. Seeam et al. (Eds.): InterSol 2023, LNICST 541, pp. 337–354, 2024.
https://doi.org/10.1007/978-3-031-51849-2_22

distributed systems are, traditionally, considered as processes having a precisely specified behavior, and thus determined. The emergence of the Internet as a platform for the implementation of these architectures tends to render this hypothesis obsolete, because it brings into play intelligent users who act according to their own strategy. This paper raises the issue of the difficulties of maintaining cooperation and the selfish behavior of peers. In [3] peer-to-peer networks are a beneficial way to offer and share services. In recent years, they have become very popular with the general public and professionals. This is because these systems are inexpensive, fault and disconnection tolerant, easy to use and scale easily (up to several million users). These advantages are due to a strong concept: the decentralization of services. Each user is both customer and provider of a service. Attached is the illustrative Fig. 1 of most important P2P applications.

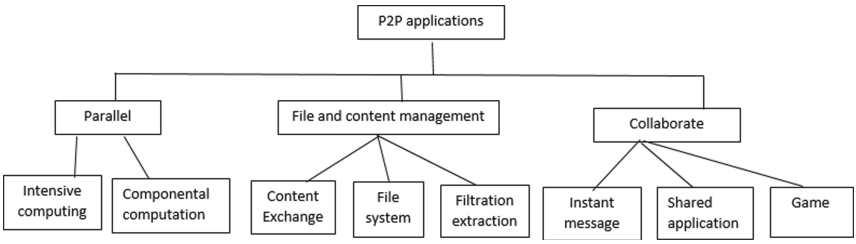


Fig. 1. P2P applications

Selfish behavior of peers [2] is also called "free-riding". Thus "free-ridingis" developing in P2P networks and negatively impacts the robustness and availability of the network. Indeed, some users use files shared by other users without sharing files themselves which would cost them bandwidth. They thus benefit from the advantages of the collective good without paying the cost. Thus, through this free riding behavior, the peers who are among the biggest contributors act as central servers, which could lead to network congestion and instability, problems that the P2P system was supposed to combat. As a result, we see a tragedy of the commons. In [2], this expression (Tragedy of common) illustrates an abusive use of common resources leading eventually to their exhaustion.

Also it is said that in order to answer the various concerns of research works were interested in the question and made it possible to develop protocols of coordination in the peer-to-peer networks.

With the aim of finding elements of answer to the problematic above posed, we leaned on the topic: "Study and simulation of the rewiring factor in an unstructured P2P network." In our review of literature in relation with [2], we saw according to Hales that one of the characteristics of the SLACER protocol is to allow a self-organization of the nodes of the system.

The objective in this document is to make a study then an implementation of the SLACER protocol which will make it possible to obtain cooperative artificial social networks based on the peer-to-peer. The SLACER protocol is a solution proposed by a European project. We will therefore investigate whether the results of the paper are valid for different variations of the rewiring parameter named W and on a wider range of networks. Our paper is structured according to the following plan:

1. First, we present the related work. This part gathers the presentation of the SLAC and SLACER protocol according to our literature reviews. We will also see some concepts and definitions, which will allow us to better understand the rest of the work;
2. In a second step we present and define the simulation environment that would be used for the tests. We present the methodology to be followed to perform the simulation on a larger network size. We discuss random sampling, then the prisoner's dilemma game and the configuration parameters required for the SLACER protocol;
3. In a third step, we present the simulation results obtained for several values of the rewiring factor W as well as the different interpretations;
4. Finally, we present ours contributions and perspectives.

2 Related Work

2.1 SLAC Protocol

In [4] the protocol **SLAC**: (Selfish Link and behavioural Adaptation to produce Cooperation) is an algorithm where each user can, and is in fact encouraged to, act selfishly. In a P2P network there are users who only download files and never share files with others. These users become leeches and drain the resources of the community. These types of users are called free loaders. We show that SLAC allows an unstructured P2P overlay network to evolve to a state where free loaders and attackers are quickly banned without asking any user to behave against their interest.

Unlike free loaders that only reduce network resources, there are also malicious users who launch active DOS attacks against P2P overlay networks. These attacks usually take the form of spreading fake files or overloading the network by flooding with fake requests. they use the flooding mechanism of repeating a message across all networks. A node that initiates the flooding sends the packet to all its direct neighbors, similarly if any network node receives the packet for the first time, it rebroadcasts it to all its neighbors. Thus, from one node to another, the packet floods the network.

SLAC works by taking advantage of a key property of the flooding-based search mechanism used in the unstructured P2P overlay network. In flooding, when a node wants to find a particular piece of data, it sends a search request to all of its neighbors on the P2P overlay network. Its neighbors then in turn

forward the search request to their neighbors, and so on. Note that this flooding-based search mechanism allows neighboring nodes to mutually control access to the rest of the network. We present below the Algorithm 1 which represents the SLAC protocol.

Inputs :
i : node i
j : node j
Ui : Utility of node i
Uj : Utility of node j
Si : strategy of node i
Li : Links of node i
Outputs: Operation
Begin:
Compare Ui and Uj
if $Ui <= Uj$ **then**
Copy Si and Li (*reproduction*)
Mutate (*with a small strategy*):
Change *strategy* (*behaviour*)
Change *neighborhood* (*links*)
end if
End

Algorithm 1: SLAC

The algorithm runs continuously in each node. A periodic utility comparison is made between two randomly chosen nodes in the node population. In [2], Utility is a numeric value that each node must calculate on the basis of the particular application domain's specifics. Also the strategy represents the behavior of the nodes at the level of an application. Either by deleting and or creating links to the nodes they know. If the utility of the latter is superior, then copy its strategy and links. Then, perform a mutation with low probability and change your strategy and links.

We try to explain the operation of the SLAC algorithm from Fig. 2.

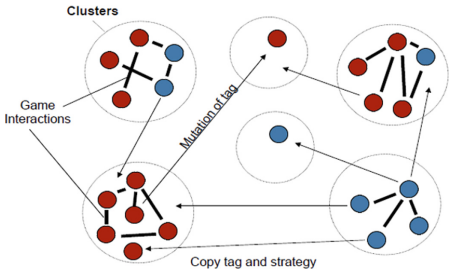


Fig. 2. SLAC operation [11]

In Fig. 2 above, the different circles represent clusters of nodes that are a collection of several nodes that communicate with each other to perform a set of operations. According to [5], tags define social rules of behavior observed in human societies. Each node belonging to a cluster or not, has its own tags. Studied by computational sociology, tags are observable marks (e.g. hairstyle, clothes), which evolve like any other artificial trait and allow to limit interactions between agents with similar tags leading to cooperative altruistic behavior. Thus, we describe our SLAC protocol operation scheme as follows:

- the different nodes in the P2P network move to find cooperative neighbors;
- This perpetual search for cooperation produces a kind of evolution of the network;
- nodes that do not cooperate in the network become isolated.

The results we can have in a duplicate and rewire rule are :

- produce a kind of «cluster selection» between clusters;
- give a functional reason for the temporal structures present in «natural» networks.

In addition the tags can undergo a mutation. According to [5], the mutation of the tags consists in a change of certain features (i.e. certain bits) and the mutation of the strategy, it consists in turning over the strategy bit (from cooperation to defection or vice versa). After the periodic comparison of utilities, whether or not this results in the copying of another node, the node resets its utility to zero.

2.2 SLACER Protocol

In [2] the SLACER algorithm is an extension of the SLAC Algorithm [7] based on the tagging approach from computational sociology that is explained in [6] which has been shown to support high levels of cooperation without the need for central control, reciprocity or other evaluation mechanisms.

Over time, nodes engage in an application task and generate an utility measure U. According to [10], in the SLACER algorithm, each node engaged in a specific application task generates a utility measure called U. Such measures are strictly related to the application domain an can be defined in very different ways according to the application's aims. For example, they could be defined as a function of download speed in a file sharing scenario; as the number of jobs processed in a distributed computing environment; as the latency and ratio of delivered packets in a routing protocol, etc. The higher the value of U, the more the node believes it is working in its target domain.

The authors in [2] state that periodically, each node i compares its performance to that of node j. chosen at random from the population of nodes. The SLACER protocol consists of four steps, which we detail in the algorithm 3, 4, 5, 6 below.

```
Active thread of node i
Inputs :
        Pi : View perimeter of node i
        Ui : Utility of node i
        Uj : Utility of node j
        i : node i
        j : node j
Outputs: Operation
Begin:
do forever
j= GetRandomNode(Pi)
if Uj <= Ui then
   CopyStatePartial(j)
   Mutate (i)
end if
End
```

Algorithm 3: Active thread of node i

```
Function CopyStatePartial of node j
Inputs :
        Ui: Utility of node i
        Uj: Utility of node j
        Si :Strategy of node i
        Sj :Strategy of node j
        Lj :Links of node j
Outputs: Operation
Begin:
Si = Sj
For each Link in Li drop Link with probability W
For each Link in Lj Li.addLink(Link)
End
```

Algorithm 4: Function CopyStatePartial

If utility Ui is less than or equal to that of Uj, node i removes each of its current links to other nodes with high probability W, and copies all the j links and adds a link to j itself. In addition i copies the strategies of j. After the copy operation with low probability M, node i adapts its strategy and then with probability MR adapts its links. The adaptation involves the application of a «mutation» operation. The SLACER algorithm is composed of four parts, mainly divided into two (02) threads and two (02) functions. Indeed, we have :

- an active thread,
- a passive thread,
- a strategy copy function,
- a mutation function where each existing link will be deleted with probability W and, a unique link to a randomly drawn node in the network will be added.

```
Function Mutate of node i
Inputs :
        Pi: View perimeter of node i
        Si : Strategy of node i
        Sj : Strategy of node j
        Li : Links of node i
        Lj : Links of node j
Outputs: Operation
Li: Links of node i
Si: Strategy of node i
Begin:
Mutate Si with probability M;
Mutate Li with probability MR;
For each Link in Lj drop Link with probability W
i.addLink(SelectRandomNode(Pi))
End
```

Algorithm 5: Function Mutate

```
Passive thread of node j
Inputs :
        Uj: Utility of node j
        Sj : Strategy of node j
        Lj : Links of node j
Outputs: Operation
Begin:
do forever
Sleep until a request received from node i
Send Uj to node i
Send Lj to node i
Send Sj to node i
End
```

Algorithm 6: Passive thread

The authors of [4] state that SLACER follows the same general approach as SLAC but is more conservative as it maintains old links rather than rewiring all links, it is an adaptation based on selfish links for cooperation outside the rewiring algorithm (SLACER). Assume that peer nodes can change their strategy. Also nodes have the ability to discover other nodes at random from the population set of nodes, compare their performance to those nodes in some way and copy their links and strategies. SLACER implements a simple local adaptation rule: nodes try to use their capabilities to selfishly increase their own performance (or utility) in a greedy environment adaptively by changing their links and strategies. They do this by copying nodes that appear to perform better and making random changes with low probability. The SLACER algorithm runs continuously in each node.

3 Methodology for Implementing the SLACER Protocol

3.1 Definition and Simulation Environment

The peersim simulator environment is released to the public under the GPL open source license. It is written in Java and consists of two simulation engines, a simplified (cycle-based) and an event-driven one. The engines are supported by many simple, extensible and pluggable components, with a flexible configuration mechanism.

In fact, in [9], the main difference between the two models (event-driven, cycle-driven) is the different management of time; in the cycle-driven model, it is given in execution cycles whereas in event-driven mode it is given in units of time. Once the unit of measurement of time (or endtime simulation) has been chosen, consistency is required each time it is necessary to insert or manipulate a time value.

3.2 Random Sampling

In Fig. 2, we assume a function that returns a random node from the entire population of nodes, regardless of the current network topology. This function cannot use the SLACER-managed network itself because it may become partitioned. In our simulations, we used the existing NEWSCAST algorithm [8] to provide this service. NEWSCAST provides exactly this function by maintaining its own scalable and robust random overlay network based on a gossip protocol; in which random gossip neighbors constantly collect their views of the network (the links they have with other nodes). This type of sampling maintains a random, fully connected topology, even under conditions of high node failure and malicious behavior. NEWCAST can be deployed to support SLACER in a modular fashion via a GetRandomNode () function invoked from SLACER.

3.3 The Prisoner's Dilemma Game

In [4], the authors of the paper explain the prisoner's dilemma problem as follows: Assuming that neither player can know in advance which move the other will make and wishes to 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, because its the highest payoff, only one can attain it in a single game. Two players interact by selecting one of the following two choices: Cooperate (C) or Default (D).

According to [1], the authors explain the prisoner's dilemma as follows: «Two suspects in a major crime are held prisoner in separate cells. There is enough evidence to charge each of them with a minor crime, but not enough to charge them with the major crime unless one of them denounces the other (treason). If both inmates remain silent, each will be charged with the lesser crime and will face a one-year prison sentence. If one (and only one) of the two denounces the other, he will be released and will testify against the other who will spend four

years in prison. If the two suspects turn each other in, each will face three years in prison.»

Since it is two players, we have four possible outcomes of the game, the players receive specific payoffs. According to [2] both players receive a reward (R) for mutual cooperation and a punishment (P) for mutual defection. However, when individuals choose different moves, the defector receives a temptation payoff (T) and the cooperator receives a sucker payoff (S). T, R, P and S are payoffs ranked from highest to lowest.

This situation can be illustrated by Table 1 which represents a matrix of gains.

Table 1. Prisoner's dilemma game

Player1/Player2	Cooperate	Default
Cooperate	**R/R**	S/T
Default	**T/S**	P/P

Table 1 illustrates a situation in which the players have an interest in cooperating. But, each player has an incentive not to do so (free riding). Furthermore, we let the nodes play probabilistic strategies where their movements were determined by a real value indicating a probability to cooperate (in this case, the mutation involved changing the real value into another one, uniformly selected at random from [0.1]).

3.4 Simulation Indicators

Simulation Cycle: This is the time period during which a node initiates activity at the application level, causing a utility update and executing a Compare Utility call.

Measuring Cooperative Connected Path (CCP): represents the ratio of pairs of nodes connected by paths composed only of cooperative nodes to all possible pairs of nodes in the network. The maximum possible value is obviously 1 [2]. A network does not have to be fully cooperative to reach the maximum value, because it is possible to find other cooperative paths to avoid faulty nodes. We will use Fig. 3 as an illustration. On the other hand, a network cannot reach the maximum value even if it is fully cooperative but also partitioned. Indeed, in this case, there are no paths between the nodes of different partitions of the network.

Giant Connected Component (GCC): It is the largest single connected component. In network theory, a giant component is a connected component of

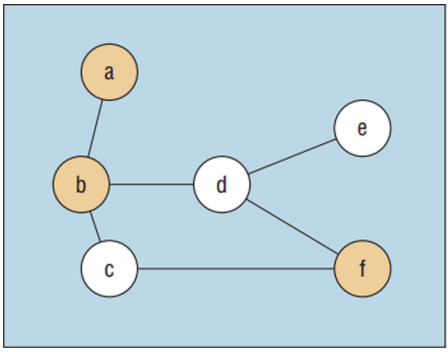


Fig. 3. CCP Measure [2]

a given random graph that contains a finite fraction of the vertices of the entire graph.

Clustering Coefficient (C): The latter is still called clustering coefficient (noted C) for a single node i, is the ratio between the number of current links between its neighbors and the total number of possible links. If Zi is the number of neighbors linked to node i and Yi the number of links between neighbors of node i, with N the total size of the network. The clustering coefficient is represented by the following formula 1:

$$C = \frac{1}{N} \sum_{i \in N} \left(\frac{2Yi}{Zi(Zi-1}} \right) \tag{1}$$

The Average Path Length (L): We define L as the shortest possible average path between all pairs of nodes. With d(i,j) the shortest distance between nodes i and j in the network. The average path length is represented by the following formula 2:

$$L = \frac{1}{N} \sum_{i \in N} \sum_{i \neq j} \left(\frac{d(i,j)}{N-1} \right) \tag{2}$$

The Average Payoff: The Prisoner's Dilemma application defines the utility value required by SLACER as the average payoff the node received from these game interactions.

3.5 The Configuration Parameters

Figure 4 represent the different parameters to configure on the configuration file to launch the SLACER protocol. The figure shows the SLACER configuration parameters (a) in this part, there are:

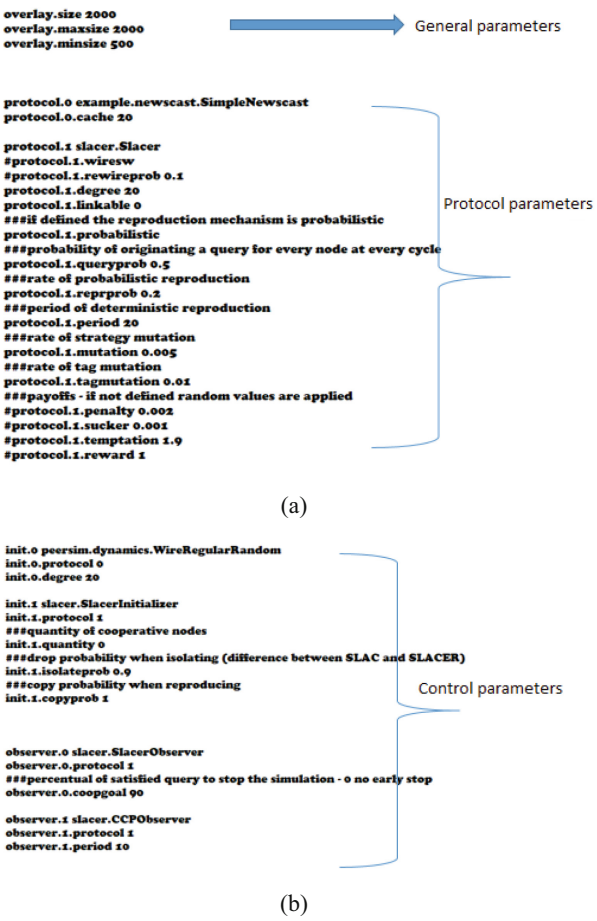


Fig. 4. Parameters of SLACER's configuration (a) Protocol parameters (b) Control parameters

1. General parameters: we define the size of the network;
2. The protocol parameters part: we define the protocols (Newscast and SLACER) then the parameters of the prisoner's dilemma (DP);
3. The control and initialization part used to define the operations that require knowledge and management of the global network, such as initializers (executed at the beginning of the simulation), dynamics (executed periodically during the simulation), and finally observers (executed periodically during the simulation).

4 Simulation Results

The objective here is to analyze the results before proposing solutions to ensure better performance.

Previous work has shown that SLACER takes 90 cycles to reach high cooperation. Compared to the high cooperation simulation cycle, in our case, for W=0.9 it takes 70 cycles to reach high cooperation.

4.1 The Simulation Parameters

In the following simulation, we set the prisoner's dilemma payoffs to:

- T = 1,9
- R = 1
- P = 0,002
- S = 0,001

Therefore, we have set the transfer rates at:

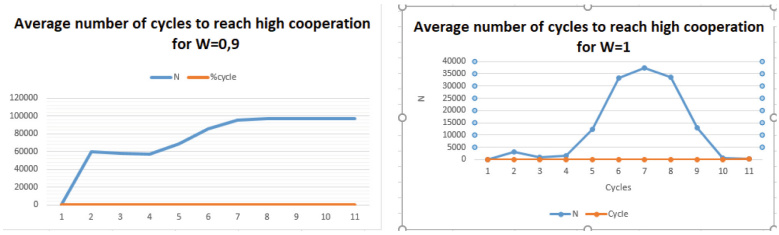
- M = 0,005
- MR = 0,01

Nodes were also allowed a maximum size of 20 neighbors. We ran a simulation with rewiring factor for W = 0.4, W = 0.9, W = 1.

4.2 Simulation Results and Interpretations

In our experiments, we initialized the entire population of nodes over a range of N = 2000 to 100000 nodes. In the initial state, all nodes were in the defection state and we connected them in a random network topology.

1. Simulation cycle
   The presentation focuses on the average number of cycles to achieve high cooperation for different values of W.
   Figures 5 represent the average number of cycles to reach high cooperation for different network sizes.
   Figure 5 bearing the letter (a). The values taken are a function of the rewiring factors W=0.9 (this rewiring factor represents the SLACER algorithm). The network takes 70 simulation cycles to reach high cooperation. In this case we can deduce that the higher the size of the network the faster the cooperation, and SLACER can thus escape a total defection.
   And W = 1 (rewiring factors represents the SLAC Algorithm) for Fig. 5 bearing the letter (b). Cooperation does not begin until cycle 50 and reaches its emergence at cycle 70. Then, as the simulation continues we observe a drop in cooperation. This means that the nodes become isolated afterwards, creating a disconnected network.
   Figures 6 represent the average number of cycles to reach high cooperation for different sizes of the network. The values taken are a function of the rewiring factor W=0.4. At this level, according to the figure, it can be observed that cooperation takes place rapidly and is maintained throughout the cycle.



(a) (b)

Fig. 5. Average number of cycles to reach high cooperation for W = 0,9 and W = 1. (a) W = 0,9. (b) W = 1.

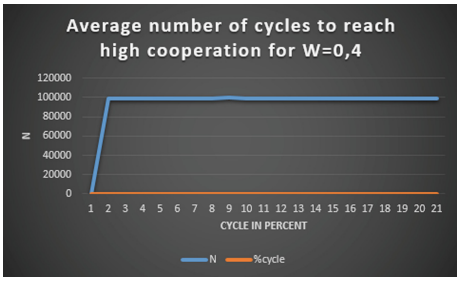


Fig. 6. Average number of cycles to reach high cooperation for W = 0,4.

2. GCC and CCP measurements
   We present the GCC and CCP measurements for W = 0.4, W = 0.9 and W = 1.
   Figures 7 represent the size of the largest component as a proportion of the network size as well as the cooperative connected path measure for W = 0.4, W = 0.9 (SLACER) and W=1 (SLAC). Each bar shows an average of 10 runs.
   For W = 0.4, we observe that almost all nodes inhabit a giant connected component (GCC) which, although containing faulty nodes, provides a cooperative route between the vast majority of its members. The high CCP indicates that the majority of nodes within the GCC are interconnected by cooperative paths.
   For W = 0.9, we can see from the figure that almost all nodes inhabit a giant connected component (GCC) that, although containing faulty nodes, provides a cooperative route between the vast majority of its members. The high CCP indicates that the vast majority of nodes within the GCC are connected by cooperative paths. This means that SLACER therefore generates cooperative ASNs (artificial social networks).
   However, for W = 1, we observe that at the beginning of the simulation the nodes inhabit a giant connected component (GCC or PGC) which decreases as the simulation continues until it becomes very negligible. This means that

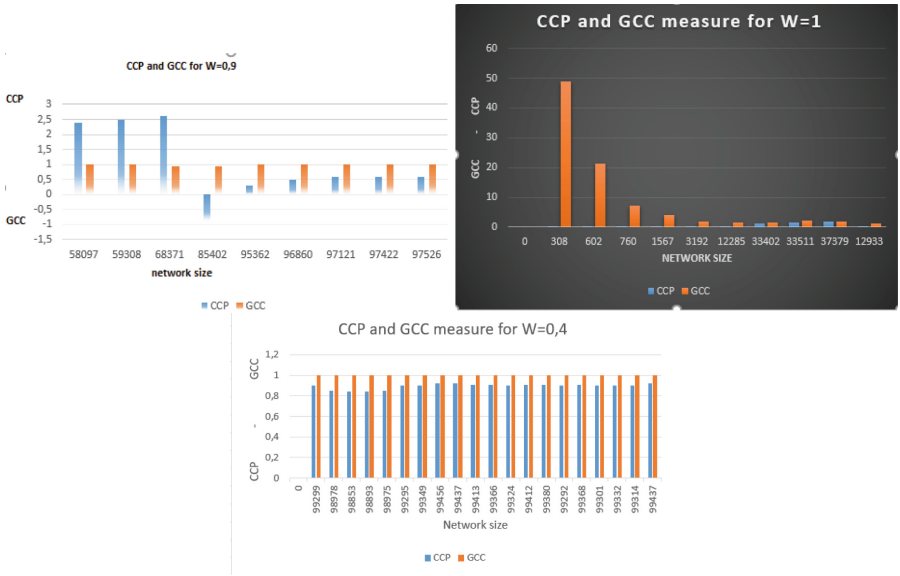
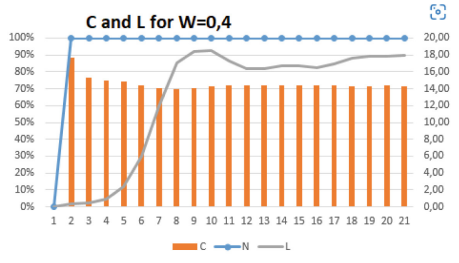


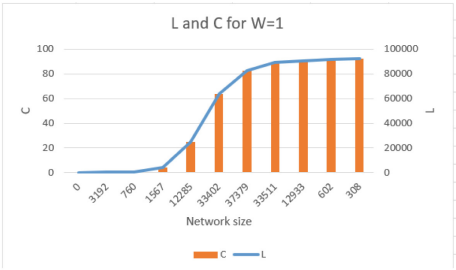
Fig. 7. GCC and CCP measure for W = 0.4, W = 0.9 and W = 1

we had few cooperative routes between the nodes. The level of the PGC being very low means that the vast majority of nodes within the GCC are not connected by cooperative paths. We can therefore say that SLAC provides a network of disconnected nodes.

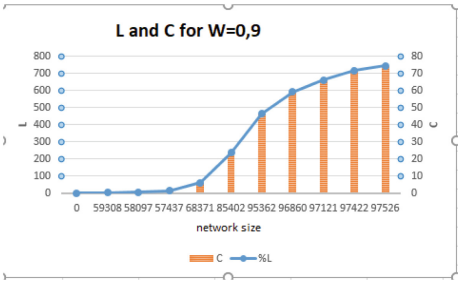
3. Average length L and clustering coefficient C measurements
   Figure 8 result representing the measure of the clustering coefficient (C) and the average path length (L) for different variations of W.
   Figure 8 with the letter (a) represents the measurement of the clustering coefficient (C) and the average path length (L) for W = 0.4. At this level we can see that while the level of C rises rapidly at the beginning of the simulation and eventually falls and finally remains more or less constant. We can therefore say here that with a W = 0.4, we witness a massive clustering of nodes at the beginning of the simulation and then an isolation of clusters of nodes that are characterized by the decrease of the clustering coefficient C. As for L, it takes more than 70 cycles to reach emergence. This means that within a cluster, the nodes are connected by jumps.
   Figure 8 with the letter (b) for W = 1 following different network sizes. Each point is an average value of 10 cycles of simulations. Each point is an average value of 10 cycles of simulations. We observe that as the simulations evolve, L evolves according to the network while C experiences some variations. This means that there is a clustering of nodes at the beginning of the simulation, however this level of clustering drops for a while causing some nodes to become isolated before gradually recovering. L being high indicates that most nodes are connected by a few hops.



(a)



(b)



(c)

Fig. 8. Average path length L and clustering coefficient C for W = 0,4, W = 0,9 and W = 1. (a) W = 0,4. (b) W = 0,9 (c) W = 1

Figure 8 with the letter (c) represents the measurement of the clustering coefficient (C) and the average path length (L) for W = 0.9 following different network sizes. Each point is an average value of 10 cycles of simulations. We observe that the level of C and L increases progressively with the size of the network. The level of C means that we are witnessing a reorganization of the network into a small world topology. L being high indicates that most of the nodes are connected by hops, so the nodes are connected by short paths.

4. Global simulation This figure below shows a series of runs for a network size ranging from 2000 to 100,000 nodes running W = 0.9

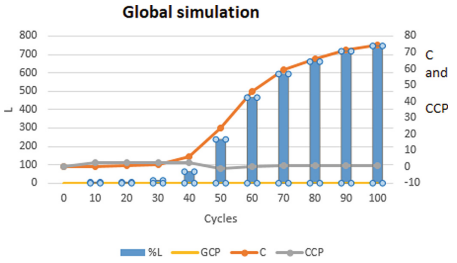


Fig. 9. Global simulation for W = 0,9

Figure 9 shows a runtime series for a network size ranging from the interval 2000 to 100,000 nodes running SLACER (W = 0.9). We can identify a number of distinct stages in the temporal evolution of the network before high cooperation is reached. First, we notice that (C) and (L) increase before the emergence of cooperation. Just after cycle 60, via a random mutation, two linked nodes become cooperative. When cooperation emerges, it spreads over the whole network reducing C to L. Once cooperation emerges, this will lead to a rapid saturation of cooperation. From this point on, we distinguish four stages: random rewiring, seed formation events, defectors will tend to disappear as nodes move to tribes with better utility. However, by defecting and acting selfishly, a node sows the seeds of its own tribe's destruction, as a node's high initial utility leads to it being surrounded by copycat defectors, reducing its gains.

5 Propositions

We believe that SLACER is a step towards very useful artificial social networks. We are also convinced that SLACER could also be applied to more realistic and useful application tasks as the SLAC algorithm being an earlier version of SLACER is applied to a P2P file sharing application and we should follow the same approach. Also we have a specific interest in producing Artificial social networks (ASN) for collective filtering of P2P viruses, spyware and spam. There are also applications that support trust and cooperative interactions between nodes in a P2P search engine.

However, although sophisticated P2P algorithms already exist, they tend to assume that all peers will act cooperatively. This is dangerous because there will always be a need for pre-existing trusted social networks to provide input, yet this is not always available or appropriate.

When dealing with complex applications, involving many types of trust and peer interaction tasks, it would be possible to create multiple instantiations of SLACER each supporting different tasks running simultaneously. In this way, each ASN would be tailored to the particular requirements of the specific task associated with it. This could be likened to the way humans form many networks around the different goals and tasks they have to accomplish in daily life.

6 Conclusion

In our paper, our main contribution lies in the simulations performed for several values of the rewiring factor W in P2P networks in order to analyze the results obtained and compare them to those of the literature. In our simulation scenario, we use a network size between 2000 and 100,000 nodes and the parameters T = 1.9, R = 1, P = 0.002, S = 0.001. We also fixed the mutation rates at M = 0.005, MR = 0.01. Nodes were also allowed to have a maximum size of 20 neighbors. While the author uses a network size between 2000 and 64000. The mutation rates at M = 0.001 and MR = 0.01. All nodes are placed in the defection state, which allows us to see if the SLACER protocol can escape total defection. This allows us to say that SLACER is robust. Previous work has shown that SLACER takes 90 cycles to reach high cooperation. Compared to the high cooperation simulation cycle, in our case, for W = 0.9 it takes 70 cycles to reach high cooperation. In this case we can deduce that the higher the network size the faster the cooperation, and thus SLACER can escape total defection.

In the future, it would be interesting to evaluate SLACER's performance by considering a study of DHT (Distributed Hash Table) systems to improve routing and search performance, as well as collaborative anti-spam or anti-spyware systems.

References

1. Anceaume, E., Gradinariu, M., Ravoaja, A.: Théorie des jeux et systemes de pairs. Publication interne- IRISA (2004)
2. Hales, D., Arteconi, S.: SLACER: a self-organizing protocol for coordination in peer-to-peer networks. J. IEEE Intell. Syst. **21**, 29–35 (2006)
3. Siebert, J., Ciarletta, L., Chevrier, V.: Impact du comportement des utilisateurs dans les réseaux pair-à-pair (P2P): modélisation et simulation multi-agents. In: 16es Journées Francophones des Systèmes Multi-Agents-JFSMA'08, pp. 129–138 (2008)
4. Hales, D., Arteconi, S.: Friends for free: self-organizing artificial social networks for trust and cooperation. arXiv preprint cs/0509037 (2005)
5. Marzolla, M.: Simulating overlay network with Peersim. http://www.moreno.marzolla.name
6. Sueur, C.: Analyse des réseaux sociaux appliquée à l'éthologie et l'écologie. publisher=Éditions Matériologiques (2015)
7. Hales, D.: From selfish nodes to cooperative networks-emergent link-based incentives in peer-to-peer networks. In: Proceedings of the Fourth International Conference on Peer-to-Peer Computing, 2004. Proceedings, pp. 151–158. IEEE (2004)
8. Jelasity, M., Guerraoui, R., Kermarrec, A.-M., van Steen, M.: The peer sampling service: experimental evaluation of unstructured gossip-based implementations. In: Jacobsen, H.-A. (ed.) Middleware 2004. LNCS, vol. 3231, pp. 79–98. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30229-2_5
9. Montresor, A., Jelasity, M.. PeerSim: a scalable P2P simulator. In: 2009 IEEE Ninth International Conference on Peer-to-Peer Computing, pp. 99–100. IEEE (2009)

10. Hales, D., Arteconi, S., Babaoglu, O.: SLACER: randomness to cooperation in peer-to-peer networks. In: 2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing, p. 5. IEEE (2005)
11. Hales, D., Arteconi, S., Babaoglu, O.: SLAC and SLACER: simple copy and rewire algorithms for trust and cooperation in P2P