SLACER: A Self-Organizing Protocol for Coordination in Peer-to-Peer Networks

David Hales and Stefano Arteconi, *University of Bologna*

Maintaining coordination in dynamic, open peer-to-peer systems is difficult. SLACER, a simple protocol, produces incentives for coordination and cooperation and self-organizes connected cooperative and robust networks.

Coordination between nodes in open distributed computer systems is a general problem that's becoming increasingly relevant as massive peer-to-peer (P2P) systems are being deployed on the Internet. A major subproblem is establishing and maintaining cooperation between nodes. The basic dilemma of cooperation is this: if no central authority ensures that nodes cooperate for the entire network's benefit, nodes will have incentives, in some scenarios, to act selfishly and thus reduce or eliminate the benefits of forming the network in the first place. For example, if all nodes in a file-sharing network only download files rather than upload them, the network is of zero value to all—nobody can download anything. This is a manifestation of the so-called commons tragedy (see the "Related Work" sidebar).

To solve this problem, we created a simple algorithm, SLACER (a *s*elfish *l*ink-based *a*daptation for *c*ooperation *e*xcluding *r*ewiring). When executed in a P2P network's nodes, SLACER self-organizes the network into a robust artificial social network (ASN) with small-world characteristics and high cooperation. SLACER is based on computational sociology's tagging approach,[1] which supports high levels of cooperation without central control, reciprocity, or other evaluation mechanisms. This approach is based on simple rules of social behavior observed in human societies.

Although we focus on cooperation here, SLACER isn't limited to cooperation applications. We can apply it more generally to coordination problems. Elsewhere, we've applied SLACER to P2P coordination problems requiring the formation of clusters of specialized nodes that coordinate their different skills to improve network level performance.[2] This distinguishes SLACER from protocols concerned purely with cooperation maintenance.

Modeling open systems

To meaningfully discuss open P2P systems, we need two kinds of distinct models: an *in-protocol model* (what a proposed protocol does) and an *out-protocol model* (how protocols themselves can be changed, either maliciously or otherwise). The in-protocol model is relatively easy to formulate and test because it is specified as an algorithm that runs on the nodes. However, the out-protocol model is essentially a social theory concerning how users behave. This is particularly relevant for understanding how a protocol's malicious variants might spread in a population of peer nodes.

In P2P protocol design, the out-protocol model is often implicit. For example, if a protocol designer just assumes that all nodes will behave cooperatively, that implies an out-protocol model in which users aren't willing to subvert the system. Alternatively, when designers test protocols on certain kinds of attacks (where nodes behave maliciously), this implies an out-protocol model with at least some malicious users.

One way to select an out-protocol model is to import one from existing disciplines. Classical game theory, for example, generally assumes that individuals act to maximize their utility, assuming that others do the same and understanding completely the possible utility payoffs of their interactions (that is, *games*). This is called *rational action*. An alternative model from evolutionary game theory assumes that individuals will copy the behavior of others who obtain higher utility.[3]

The out-protocol model we present here follows neither of these assumptions (although, interestingly, our in-protocol imports aspects of the evolutionary approach). This approach is an inversion of other approaches. Our out-protocol model assumes a less

1541-1672/06/$20.00 © 2006 IEEE
Published by the IEEE Computer Society

Related Work with Application to P2P Network Cooperation

Given the pervasive nature of commons tragedies (see the main article), researchers have proposed several solutions that apply to open peer-to-peer systems without central control.

One broad approach is based on reciprocity—nodes are rewarded in the future for being cooperative and are punished for being selfish. Much research has been done in evolutionary game theory to assess these approaches' effectiveness. Robert Axelrod showed that the simple tit-for-tat strategy (individuals store the last interaction made by those they interact with and return the same in the future) often sustains cooperation in the Prisoner's Dilemma game.[1] However, tit-for-tat requires that the same individuals interact regularly. Other work shows how indirect reciprocity could obviate the need for the same individuals to repeatedly interact by allowing the spread of *image information*.[2] If third parties can observe interactions, they can form an image without direct interaction; they can observe and later punish a selfish individual.

In very dynamic P2P systems, where many nodes are strangers to everyone, these approaches don't directly apply. One alternative is for all nodes to have access to a shared history of each node's behavior and to collectively adapt their behavior toward strangers, such that the entire system becomes less cooperative to strangers if many strangers behave selfishly.[3] This approach works when node turnover is low and nodes can maintain a reliable shared-data structure. However, maintaining a shared history increases overhead because each node must record and report interactions.

Our approach, SLACER (a *s*elfish *l*ink-based *a*daptation for *c*ooperation *e*xcluding *r*ewiring), is a protocol that can give acceptable performance levels without storing or communicating a shared history. SLACER sustains cooperation among selfish nodes when all nodes are strangers. We tested SLACER's ability to produce cooperative, connected artificial social networks (ASNs) by having nodes play Prisoner's Dilemma and variants. In Prisoner's Dilemma, each node has an incentive to act selfishly, but collectively, cooperation is the best outcome. The game captures the tension between individual and collective interest. SLACER produced high-trust ASNs that connected the entire node population with almost all nodes reachable from other nodes via chains of cooperative nodes. The ASNs were robust and scalable even though nodes acted selfishly.

References

1. R. Axelrod, *The Evolution of Cooperation*, Basic Books, 1984.
2. M. Nowak and K. Sigmund, "Evolution of Indirect Reciprocity by Image Scoring," *Nature*, vol. 393, 1998, pp. 573–577.
3. M. Feldman et al., "Robust Incentive Techniques for Peer-to-Peer Networks," *Proc. 5th ACM Conf. Electronic Commerce* (EC 04), ACM Press, 2004, pp. 102–111.

selfish environment than either the classical or evolutionary approach assumes. This assumption relates to open P2P environments' specific nature and, ironically, to certain kinds of highly selfish user behavior. Open protocols (that is, client software) that work well (thanks to the existing cooperative network) become distributed, but malicious innovations of those protocols (maliciously hacked clients) don't. Modifying a protocol is costly and requires detailed technical knowledge. Individuals who hack clients for individual benefit rather than collective benefit likely know how this would affect the entire network if everyone used the client. So, if malicious hackers want to benefit from their clients, they shouldn't spread them widely. The converse is true for modifications that would improve the network's collective performance. Of course, this argument doesn't hold if the hacker's goal is purely malicious—to destroy the network at any cost. However, a truly open system will always be vulnerable, in some way, to such attacks.

It's still an open issue what kind of out-protocol model best matches user behavior in given contexts. So, P2P researchers select models based on hunches, anecdote, or tradition. Empirical work can help ground and validate model selection in the future.

P2P overlay networks

Our target infrastructures for the SLACER algorithm are unstructured P2P overlay networks. Such networks have a population of nodes—typically, processes situated in a physical network—that maintain symbolic links to other nodes (often called *neighbors*). P2P applications such as Skype (www.skype.com) or BitTorrent (www.bittorrent.com) implement these to provide services.

A valuable property of the overlay network abstraction is that rewiring nodes or changing the network's topology is a logical process in which nodes simply drop, copy, or exchange symbolic links. It's therefore feasible to maintain highly dynamic network topologies at the overlay layer.

The SLACER algorithm

Our algorithm follows a link-based incentive approach.[4] That is, nodes make and break links in the network to minimize the effects of others nodes' selfish behavior. So, the topology itself reflects a network of cooperation.

We assume that peer nodes can change their strategy (that is, change how they behave at the application level) and drop and make links to nodes they know. We also assume that a node can discover other nodes randomly from the entire population, compare its performance against those nodes, and copy their links and strategies. (For a discussion of how to deal with malicious noise when nodes exchange information, see this article's conclusion.) SLACER implements a simple local adaptation rule: Nodes can selfishly increase their own performance (or utility) in a greedy and adaptive way by changing their links and strategy. They do this by copying nodes that appear to be performing better and by making randomized changes with low probability.

Figure 1 shows the pseudocode. Over time, nodes engage in some application task and generate some measure of utility U. U is a numeric value that each node must calculate on the basis of the particular application domain's specifics. For example, this might be the files downloaded, jobs processed, or an inverse measure of spyware infections over some period. The higher the value of U, the better the node believes it's performing in its target domain.

Periodically, a node i compares its performance against another node j, randomly selected from the population. If $U_i \leq U_j$, i drops its current links to other nodes with high probability W, copies all j links, and adds a link to j. Additionally, i then copies j's strategy; the strategy codes application-

level behavior (and could be implemented by mobile code). After such a copy operation occurs, i adapts its strategy (with low probability M) and adapts its links (with probability MR). Adaptation involves applying a mutation operation, and link mutation involves removing each existing link with high probability W and adding a single link to a node randomly drawn from the network. Strategy mutation involves changing application behavior with low probability M; the application domain dictates the specifics of strategy mutation. After the periodic utility comparison, the node resets its utility to zero.

Each node is limited to a maximum number of links or neighbors (its *view size*). If any SLACER operation causes a node to require an additional neighbor above this limit, SLACER removes a randomly selected existing link to make space for the new link. Links are always undirected and symmetrical; if i links to j, j must also maintain a link to i (and, conversely, if i breaks a link to j, j also breaks its link to i).

Previous tag models (on which SLACER is based[5]) have indicated that the mutation rate applied to the links must be significantly higher than that applied to the strategy—by about one order of magnitude, so $MR >> M$.

When applied in a suitably large population, the algorithm follows an evolutionary process in which high-utility nodes replace low-utility nodes. However, selfish behavior isn't dominant, as we might intuitively expect, because a form of social incentive mechanism results from the emergent network topology. Although individual nodes favor high-utility, antisocial strategies, these strategies don't dominate the population. The topology therefore guides the adaptation of the strategy away from antisocial, selfish behavior.

In figure 1, we assume a function that returns a random node from the entire node population irrespective of the current network topology. This function can't use the network that SLACER maintains because it can become partitioned. In our simulations, we used the Newscast algorithm[6] to provide this service. Newscast maintains a random, fully connected overlay topology even with high node failure and malicious behavior, so we could deploy it to modularly support SLACER. SLACER invokes a call to the lower Newscast layer when it requires a randomly sampled node from the population. Newscast is based on a gossip protocol; each node has a bounded network view composed by immediate neighbors' node descriptors (that is, time-stamped logical links). Each Newscast node periodically exchanges its view with that of a randomly selected neighbor; Newscast merges the two views and keeps the newest links. In this way, a random overlay topology is maintained in a lightweight fashion, and robustness to nodes joining or leaving the system is achieved through descriptors' aging.

```
periodically for node i:
    select a random node j from the population
    if utility U_i ≤ utility U_j
        copy strategy from j
        drop each link from i with prob(W)
        copy each link from j
        link to j
        with prob(M) mutate strategy of i
        with prob(MR) mutate links of i
    reset utility U ← 0;
```

Figure 1. The SLACER protocol runs continuously in each node. Each node maintains a utility value that the application level activity sets.

The Prisoner's Dilemma

The two-player, single-round Prisoner's Dilemma game captures a situation where a contradiction exists between self-benefit and the collective benefit. Two players interact by choosing to cooperate (C) or defect (D). For the game's four possible outcomes, players receive specified payoffs. Both players receive a reward payoff (R) for mutual cooperation and a punishment payoff (P) for mutual defection. However, when individuals select different moves, the defector receives a temptation payoff (T) and the cooperator receives a sucker payoff (S) (see figure 2a). We assume that neither player can know in advance which move the other will make and that players wish to maximize their own payoff. The dilemma is evident in the payoff ranking $T > R > P > S$ and the constraint $2R > T + S$. Although both players would prefer T, the highest payoff, only one can attain it in a single game. No player wants S because it's the lowest payoff. No matter what the other player does, a player always gets a higher score by defecting than cooperating. D is therefore the dominant strategy; an ideally rational player would always choose D.

So, the dilemma is that if both players cooperate, they are jointly better off than if they both defect; however, selfish players have incentives to select mutual defection. Prisoner's Dilemma is a minimal test that captures a range of possible application tasks in which nodes must establish cooperation and trust with their neighbors without central authority or external mechanisms that enforce it.

Application-level behavior involves each node playing Prisoner's Dilemma with randomly selected neighbors in the SLACER constructed network. The Prisoner's Dilemma application then sets the utility value that SLACER requires as the average payoff a node receives from game interactions. The SLACER algorithm then adapts the links and strategy of the nodes as we discussed earlier.

We also tested variants of the Prisoner's Dilemma payoffs (see figure 2b). We varied the T payoff over a number of values strictly inside the interval $[2R..R]$ (when $T > 2R$ or $T < R$, there is no longer a dilemma). We also tried a generalized Prisoner's Dilemma[3] to model client and server interactions where payoffs are asymmetric (see figure 2c). Additionally, we let nodes play probabilistic

Player 1

Player 2	C	D
C	*R*/*R*	*S*/*T*
D	*T*/*S*	*P*/*P*

(a)

	C	D
C	1/1	0/1.9
D	1.9/0	0 / 0

(b)

Server

Client	C	D
C	7/–1	0/0
D	7/–1	0/0

(c)

Figure 2. The payoff matrix shows (a) the Prisoner's Dilemma payoff structure, (b) the values used, and (c) alternative values from an asymmetric generalized Prisoner's Dilemma where a single player determines payoff values.[3] (C means that the player cooperates, and D means that the player defects. *T*, *R*, *P*, and *S* are payoffs ranked from highest to lowest.)

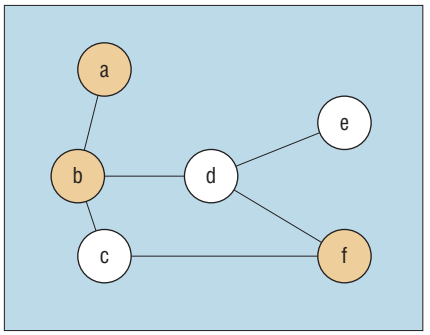

Figure 3. A small network's cooperatively connected path measure. The shaded nodes are defector nodes. Because the population has six nodes, the total number of node pairs is 15. However, only the node pairs *{a, b}*, *{b, c}*, *{b, d}*, *{b, e}*, *{b, f}*, *{c, f}*, *{d, e}*, *{d, f}*, and *{e, f}* have cooperatively connected paths, so the CCP measure is 9/15 = 0.6.

strategies where their moves were determined by a real value indicating a probability to cooperate (in this case, mutation involved changing the real value to another, uniformly selected at random from [0..1]). In each case, we found no major differences in our results.

Measuring cooperatively connected paths

Our goal in designing SLACER is to self-organize artificial social networks with desirable properties of human social networks, including

- short paths between all members,
- high cooperation between neighbors, and
- cooperation paths between members not directly connected.

To evaluate the ASNs' quality, we measured (over the entire node population) the average path length, proportion of cooperator nodes, and proportion of nodes linked via chains of cooperators. For the third option, we introduced a cooperatively connected path measure that quantifies an ASN's cooperative connectedness.

We assume a population of nodes in which each node is in one of two states (C or D). We define the CCP measure as the proportion of all possible node pairs that are linked directly or that have at least one cooperative path between them (see figure 3).

In a connected network in which all nodes are cooperators, the CCP measure would obviously be 1. However, even with several defector nodes inhabiting a network, the CCP can still be 1, as long as the defectors' location didn't obstruct other cooperative routes between pairs of nodes—that is, alternative cooperative routes existed around the defectors. So, the CCP measure is determined by the network's topology and the strategy distribution over nodes in that topology. By combining the CCP measure with other measures, we get an idea of the ASN's strategic topography. For example, in a connected network with a high proportion of cooperating nodes and low CCP, we would conclude that a few noncooperative nodes occupied positions in the network that let them block numerous unique cooperative paths between nodes. Because our design goal for SLACER is to generate connected networks of cooperative nodes, our ideal outcome would be an ASN with the CCP at or close to the maximum of 1.

Simulation specifics

We performed simulation experiments using two independent implementations. (For an implementation in the PeerSim environment or the system itself, see http://peersim.sourceforge.net. Recent research has demonstrated how to migrate PeerSim implementations to prototype implementations for real-world testing.[6])

One time cycle

When nodes engage in their application tasks (that is, they play Prisoner's Dilemma with randomly selected neighbors), they periodically initiate the Compare Utility activity in SLACER's active thread. We programmed the simulations so that, on average, over one cycle, each node initiates some application-level activity, causing a utility update and executing one Compare Utility call. Our simulations were both semiasynchronous and fully asynchronous.

In one time cycle, the simulation selects $10N$ nodes from the population randomly with replacement (where N is the total population size). When a node is selected, it chooses a neighbor at random and plays a single round of Prisoner's Dilemma. If a selected node has no neighbors, it links to a randomly selected node so that they can play a game. Both nodes play the game, executing the strategy their current state indicates (C or D). When nodes compare utility in the SLACER algorithm, the algorithm uses the average utility over the total games played rather than the absolute utility.

Two simulation variants

In the semiasynchronous simulations, after nodes had played their Prisoner's Dilemma games, the simulation randomly selected N nodes with replacement. They executed SLACER (see figure 1), starting a new cycle. In the fully asynchronous version, the simulation executed SLACER probabilistically for each node after each game (with probability 0.1).

Here we're modeling the notion that utility updates for some application tasks might be instantaneous and asynchronous. For example, neighbor nodes might provide requested resources immediately in some application tasks. However, in others, rewards might be delayed and synchronized over a large set of nodes, especially when several nodes must coordinate to achieve a collective task before receiving their rewards (for example, in a P2P search engine pooling its page rankings). Obviously, the application task's specifics could dictate many possible schemes; however, for the purposes of our initial Prisoner's Dilemma test application, we considered these two.

Parameter settings

For the results we present here, the Prisoner's Dilemma payoffs were set to $T = 1.9$, $R = 1$, $P = 0$, and $S = 0$. We selected T as a large value less than $2R$. Although Prisoner's Dilemma defines $P > S$, we obtained no major difference in our results when P's value was small. We obtained similar results over various parameterizations of the payoff values. We set the mutation rates to $M = 0.001$ and $MR = 0.01$. Again, over various values of M and MR, in the same order of magnitude, we obtained similar results. The nodes' maximum view size was 20, so each node could link to 20 neighbors maximum. We ran the simulations with the rewire probability $W = 0.9$ over a range of network sizes from $N = 2{,}000$ to 64,000. We selected $W = 0.9$, which let the nodes keep two old links, on average, when they moved in the network by copying another node's links.

Message overheads

In a simulation cycle, SLACER produces on the order of $O(N)$ messages of overhead (we assume that a single message would fit into one packet of the underlying network infrastructure). Compared with a shared-history approach,[3] the message overhead would be on the order of $O(N(\log N))$, based on a typical distributed-hash-table approach. This assumes that the number of interactions between nodes

increases linearly with N, as our simulation cycle specifies. However, the shared-history approach deals with whitewashing attacks and so buys increased robustness.

Simulation results

In our simulation experiments, we set the entire node population to defect and connected nodes in a random network topology. Any initial network topology, even fully disconnected, had no significant effect on the simulations' cooperative outcomes. However, if we started our networks in a small-world configuration (or fully disconnected), high levels of cooperation were produced earlier (in approximately half as long). So, of the topologies we tested, random was the worst case.

Setting all nodes to defect tested whether SLACER could escape from full defection, which made us confident of the algorithm's robustness to recover from a catastrophic cooperation failure. When we set the nodes randomly to defect or cooperate (assigning 50 percent of the population to initially cooperate), high cooperation occurred much earlier (in approximately one-quarter of the cycles, compared to when all nodes defected). Similarly, if we reduced the number of nodes that initially cooperated, high cooperation took longer (assigning 10 percent to initially cooperate, the number of cycles was approximately halved). Because these results are for populations that began with all nodes defecting, this gives the worst case.

We measured how long it took for high levels of cooperation to emerge, and we stopped when 98 percent of the nodes were cooperative. We found that cooperation and topology measures were metastabilized after this point, randomly oscillating around mean values. Complete stability isn't possible because nodes are always moving and changing strategies in the network.

When nodes reached high cooperation, we measured the entire population's CCP, average path length, clustering coefficient, and largest single connected component. For each experiment, we performed 10 runs with identical parameters and different pseudo-random-number-generator seeds. Our analysis took into account the averages and variances over these runs.

Cycles to high cooperation

Figure 4a shows the number of cycles to high cooperation from an initial population of all defector nodes. SLACER takes fewer than 90 cycles to attain high cooperation

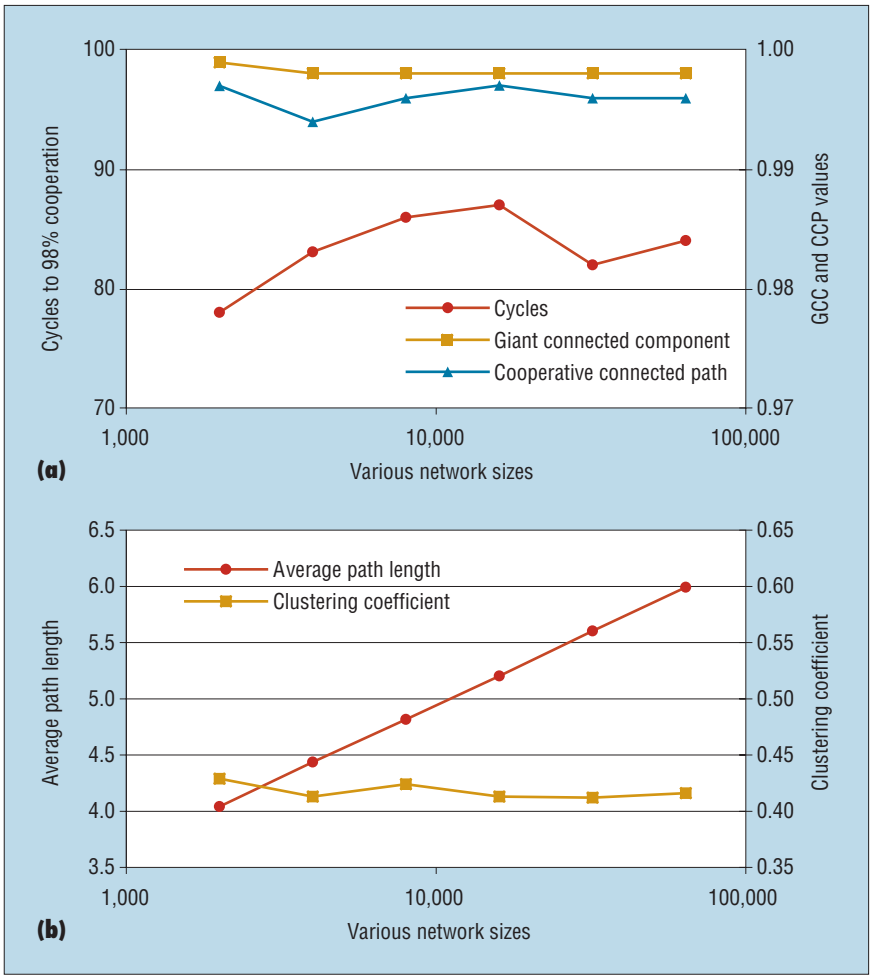

Figure 4. We examined (a) the average number of cycles to reach high cooperation, the largest component's size as a proportion of population size (the giant connected component), and the cooperative connected path measure for various network sizes; and (b) the clustering coefficient and average path length after high cooperation emerges for different network sizes. (For both charts, each point is an average of 10 runs.)

over different network sizes. An existence proof shows that the algorithm can escape all defection quickly. We obtained similar results when we used the asymmetric payoffs shown in figure 2c.

Figure 4a shows the size of the largest connected component and the CCP value for various network sizes after high cooperation emerges. We obtained high values for both CCP and the largest component size. Almost all nodes inhabit a giant connected component (GCC) that, although it contains some defecting nodes, provides cooperative routes between the large majorities of its members. SLACER therefore generates cooperative ASNs. Importantly, the CCP scales well because as N increases, the CCP doesn't decrease. There appears to be no significant scaling cost in time to high cooperation—that is, as network size increases, time to cooperation doesn't increase.

Basic topological features

We measure the networks' average clustering coefficient (c) and average path length (L) after the network has achieved high cooperation. The clustering coefficient for a single node i is the ratio of the number of current links between its neighbors to the total number of possible links. If z_i is the number of neighbors linked to node i, and y_i is the number of links between neighbors of node i,

$$c = \frac{1}{N}\sum_{i \in N} \frac{2y_i}{z_i(z_i - 1)}$$

We define L as the average shortest possible path between all pairs of nodes. If $d(i, j)$

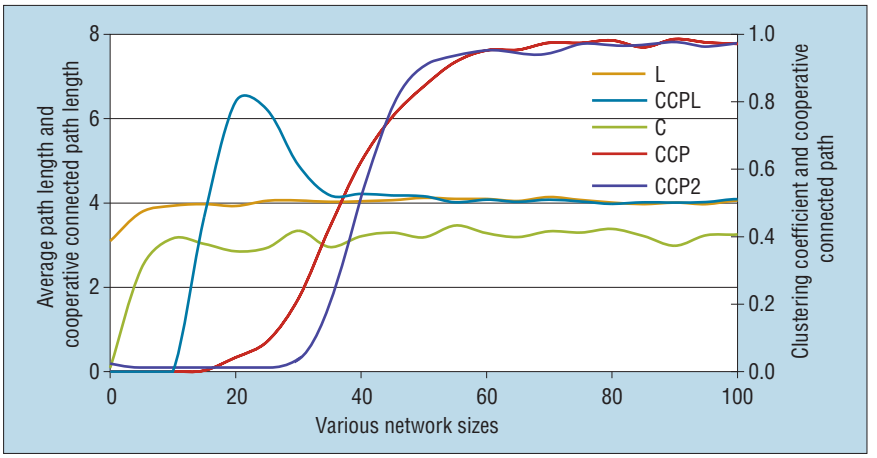

Figure 5. A typical individual simulation run for a 2,000-node network. CCP2 shows the cooperative connected paths from a different run when we set the payoffs to the asymmetric values in figure 2c. The other values for this run (not shown) follow similar paths as those shown here for the symmetric payoffs.

indicates the shortest distance between nodes i and j in the network,

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

Figure 4b shows that the networks have a small-world-like topology, because c is relatively high (compared to a random network) yet L is low—meaning that most nodes are connected by only a few hops. L scales-up log-linearly, indicating that even in very large networks, most nodes are connected by short paths. The networks' degree distributions appear linear, with approximately 10 percent of nodes having the maximum number of links and almost no nodes holding zero links. Many nodes have many links, which suggests that the networks are more robust than those displaying power-law distributions, giving a scale-free topology where only a few hub nodes have many links.

Typical evolution

Figure 5 shows a time series of a typical run for a 2,000-node network executing SLACER. We can identify several distinct stages in the network's time evolution before the network reaches high cooperation. First, c increases rapidly and L increases before cooperation emerges. This results from a randomized rewiring process, because all utilities are identical. This is sufficient to create the c and L values we find throughout the run. Just before cycle 20, via random mutation, two linked nodes become cooperative—we call this a *seed tribe*. Rather than growing locally, this seed tribe explodes, creating a kind of sparse cooperative backbone over the entire network. This spreads cooperation quickly, leading to rapid predominance of cooperation. Before seed tribe formation, nodes that behave cooperatively (via mutation) are punished heavily, gaining the S payoff as defector nodes exploit them.

We observed four stages—random rewire, seed formation, seed explosion, and saturation—in all the runs. Our previous work discusses in detail this process through selection between clusters or tribes.[5] Essentially, if tribes' organization gives their members high utility, they will tend to recruit more members. Tribes that become infiltrated with defectors will tend to die out because nodes will move to the tribes that offer better utility. Ironically, by defecting and acting selfishly, a node sows the seeds of its own tribe's destruction—the defecting node's initial high utility leads to it becoming surrounded by copycat defectors, reducing its payoffs.

Robustness to churn

We also subjected the ASN that SLACER produced to robustness tests by introducing various amounts of churn, where old nodes leave and new nodes join the network. In these experiments, we reset randomly selected nodes to defect states over various intervals of cycles. We found that even when 50 percent of nodes were replaced at one cycle, high cooperation, the CCP, and the topology structures we previously observed quickly reformed within a few cycles. We expected this because SLACER incorporates noise in the form of mutation to both links and strategies, driving its evolutionary dynamics, and can quickly recover from states of complete defection and link disconnection.

Different rewire values

We also experimented with different values of W. We found that when W was higher than 0.9, the CCP started to fall off, producing networks that were highly disconnected—a kind of extreme tribalism. However, when we reduced W below 0.9, the nodes' cooperation and the clustering coefficient began to fall. So, when $W = 0.7$, we didn't achieve cooperation levels above 90 percent of the nodes (with mean values oscillating approximately 80 percent and c at approximately 0.25). For $W = 0.5$, cooperation levels never got above 70 percent, oscillating widely approximately 60 percent, with c below 0.2. This indicates a trade-off; we need high W to get high cooperation (because this creates the tribes that drive the process), but if W is too high, we get extreme tribalism (that is, a disconnected network). Varying W, therefore, controls the strength of tribalism or cliquishness.

Probabilistic strategies, asymmetric payoffs, and specialization

We experimented with various payoffs and strategies for Prisoner's Dilemma. Probabilistic strategies (where each node stores a probability to cooperate rather than a binary flag) produced high cooperation, with most nodes selecting cooperation with a probability of 1. (This is similar to previous work, where we used a SLACER-like algorithm to adapt a generosity ratio in a simulated P2P file-sharing system.[5]) We also used the asymmetric payoffs for the generalized Prisoner's Dilemma (see figure 2c). We again found that the network obtained high cooperation with no major differences to the standard Prisoner's Dilemma case in the overall results. We applied SLACER to more complex task scenarios that require nodes to form tribes of specialists (that is, clusters of connected nodes with distinct roles) working together to attain high utility. Here we found not only cooperation and altruism but also role differentiation that was spontaneously self-organized in tribes.[2]

Whitewashing nodes

Whitewashing is a malicious attack strategy in which a node exploits zero-cost identities by continually changing its identity to avoid the consequences of malicious behavior. We tested this strategy by running experiments in which some proportion of nodes

always defected and then immediately changed identity (for example, by leaving and then reentering the network and linking to randomly selected nodes). We found that cooperation degrades gracefully in the presence of such nodes—as the number of whitewashing nodes increases, the total amount of cooperation decreases in approximately the same proportion. So, even with 20 percent whitewashing nodes, the network sustains 80 percent cooperative nodes.

We assumed that whitewashing nodes wouldn't respond to other nodes' requests for utility, strategy, and links because it's not in their interest to do so. If whitewashing nodes communicate these values, this degrades performance much further, with approximately twice the impact (20 percent of whitewashing nodes sustaining only 40 percent cooperative nodes). In each case, we assume that the whitewashing nodes don't communicate or copy the whitewashing strategy in the SLACER protocol; it's not in their interest to do so if they wish to maximize their payoff. However, if a whitewashing node did communicate the whitewashing strategy, this would destroy cooperation in the network. (We also ran experiments where nodes lied about utility and links to other nodes to further maximize their utility, and we found similar results.)

Of course, even if whitewashing strategies didn't spread in the SLACER protocol, they could spread by an out-protocol means, such as friends communicating a hacked whitewashing client. As we've shown, even under these conditions the system can tolerate numerous whitewashing nodes without breaking down. However, it would be in the system exploiters' interest to limit the spread of such a hacked client. For example, the deployed BitTorrent system operates well despite being wide open to whitewashing.

So, although SLACER is vulnerable to whitewashing, this might be tolerable given our assumptions about user behavior. Of course, this is debatable, and in a highly hostile environment, our assumptions wouldn't hold.

SLACER is a step toward potentially very useful lightweight P2P protocols that self-organize and sustain coordinated, connected networks without a shared-history system. Because all nodes are essentially strangers, the system benefits from low overhead and ease of implementation without maintaining distributed data structures for shared histories or having to log interactions.

SLACER imports an evolutionary dynamic into the protocol itself such that application-level variants (such as the strategy, which we could implement as mobile code) are automatically copied between nodes to improve performance. This produces a self-organizing, group-like selective process that supports both general cooperation and other group-functional coordination behaviors—for example, specialization in a cluster of nodes,[2] where complementary resources (nodes with complementary functions or skills) are brought together cooperatively to solve a collective problem.

Although whitewashing nodes can exploit the protocol, our out-protocol assumptions suppose that they don't spread. The protocol shows a graceful degradation of performance in proportion to the number of whitewasher nodes present. The protocol is also vulnerable to other attacks, such as Sybil, where one node accumulates many links by adopting multiple identities. However, for a network of millions of nodes, such attacks would require huge resources.

We aimed to produce a simple generalizable coordination protocol that scales well and functions under reasonable attack assumptions rather than a watertight solution that would introduce overheads. SLACER produces networks with many desirable properties of human social networks. Where existing protocols require human social networks as input,[7,8] you might be able to use SLACER instead. This would require that the applications supply suitable utilities and that our out-protocol assumptions hold. ■

Acknowledgments

The EU's 6th Framework Program partially supported this work under contract 001907 (DELIS).

We thank Ozalp Babaoglu, Mark Jelasity, Alberto Montresor, and Simon Patarin from the University of Bologna's Computer Science Department for perceptive discussions, observations, and pointers concerning peer-to-peer systems. We also thank Andrea Marcozzi and Gian Paolo Jesi, who produced an initial implementation of SLAC using Newscast in the PeerSim environment. We particularly thank the anonymous referees of earlier drafts of our article for their diligence in identifying errors and omissions.

References

1. R.L. Riolo, M.D. Cohen, and R. Axelrod, "Evolution of Cooperation without Reciprocity," *Nature*, vol. 414, 2001, pp. 441–443.
2. D. Hales, "Choose Your Tribe! Evolution at the Next Level in a Peer-to-Peer Network," *Proc. 3rd Workshop Eng. Self-Organizing Applications* (EOSA 05), LNCS 3910, Springer, 2006, pp. 61–76.
3. M. Feldman et al., "Robust Incentive Techniques for Peer-to-Peer Networks," *Proc. 5th ACM Conf. Electronic Commerce* (EC 04), ACM Press, 2004, pp. 102–111.
4. Q. Sun and H. Garcia-Molina, "SLIC: A Selfish Link-Based Incentive Mechanism for Unstructured Peer-to-Peer Networks," *Proc. 24th IEEE Int'l Conf. Distributed Computing Systems*, IEEE CS Press, 2004, pp. 506–515.
5. D. Hales and B. Edmonds, "Applying a Socially Inspired Technique (Tags) to Improve Cooperation in P2P Networks," *IEEE Trans. Systems, Man, and Cybernetics—Part A: Systems and Humans*, vol. 35, no. 3, 2005, pp. 385–395.
6. M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-Based Aggregation in Large Dynamic Networks," *ACM Trans. Computer Systems*, vol. 23, no. 1, 2005, pp. 219–252.
7. J.S. Kong et al., *Let Your CyberAlter Ego Share Information and Manage Spam*, Los Alamos Nat'l Laboratory, 2005; http://xxx.lanl.gov/abs/physics/0504026.
8. S. Marti, P. Ganesan, and H. Garcia-Molina, "DHT Routing Using Social Links," *Peer-to-Peer Systems III: 3rd Int'l Workshop* (IPTPS 04), LNCS 3279, Springer, 2005, pp. 100–111.

The Authors

David Hales is a postdoctoral researcher at the University of Bologna's Department of Computer Science. His research interests include agent-based social simulation, self-organizing software, applying social theories to computational systems, and peer-to-peer systems. He received his PhD in computer science from Essex University. He's an elected officer and member of the European Social Simulation Association. Contact him at the Dept. of Computer Science, Univ. of Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy; dave@davidhales.com; www.davidhales.com.

Stefano Arteconi is a PhD student in computer science at the University of Bologna. His research interests include peer-to-peer networks and complex systems, particularly in applying biological and sociological techniques to distributed systems. He received his MSc in computer science from the University of Bologna. Contact him in care of David Hales, Dept. of Computer Science, Univ. of Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italy; arteconi@cs.unibo.it.