# Towards Automatic Social Bootstrapping of Peer-to-Peer Protocols *

David Hales
University of Bologna
hales@cs.unibo.it

Ozalp Babaoglu
University of Bologna
babaoglu@cs.unibo.it

## ABSTRACT

Current peer-to-peer systems rely on user intervention to install and update the software (protocols) that run on each node. We propose another direction where protocols are dynamically and automatically rolled-out over peers, with the peers themselves selecting those that are beneficial and rejecting those which are not. To achieve this, we argue that, new protocols should be "injected" live into a running P2P system, with peers themselves replicating them. This requires that peers select "socially beneficial" protocols even though they need to base this on their own individual performance evaluations. What we are proposing can be seen as a meta-protocol, which we call *Automatic Social Bootstrapping*, that intelligently selects and replicates those protocols that are for the social good — that is, maximize the average utility of the entire population. We sketch an outline of the protocol and present some initial high-level simulation results. Finally we identify several open issues that need to be addressed in order to further develop the approach.

## 1. INTRODUCTION

There is a trend towards distributed social software. Often based on peer-to-peer (P2P) protocols, such systems harness the social behavior of a community of users: for example sharing bandwidth or data to achieve individual user goals, downloading a particular media file or establishing real-time audio contact between peers.

Currently, for social software to "take-off" there needs to be out-of-band publicity and user interventions. This means millions of users need to locate, download and update popular P2P clients such as BitTorrent [13], Skype [14] or eDonkey [15]. But at the machine level, these manual distribution methods are painfully slow and inefficient, relying on users to communicate, coordinate and act. Also this could be holding back the deployment of services that can't create their own "buzz" — that don't stimulate the kind of publicity necessary for successful massive deployment.

Many currently successful systems appear to have gained popularity due to the social pressure that they implicitly produce for users to join. For example, a user of Skype (a free voice-over-IP service) will naturally request their regular contacts to install the service so they can make free telephone calls. Others have discussed how seductive "psychological", rather than economic benefits, are sometimes employed [12].

*This work was partially supported by the EU within the 6th Framework Programme under contract 001907 (DELIS).

In addition to recruiting users to download and maintain clients, systems need to keep users in check. Incentives are needed to ensure that individual users continue to contribute to the social good. For example, if file sharing systems allow users too much latitude to "leech" — only downloading and not uploading — then the knock-on effect will be to degrade the overall performance of the system. Various mechanisms are employed such as tit-for-tat-like strategies (in BitTorrent) or high-levels of code obfuscation in closed protocols (in Skype).

For these kinds of social software systems, the *social bootstrapping process* — getting the protocol running on many nodes of a network — is still essentially a manual procedure. It is as though we are back to the early days of mainframes where to get an application running required direct operator intervention over several stages. In some sense, the predicament here is worse since it requires many hundreds or thousands of individual user interventions (often including reboots of machines). This is highly inefficient and tiresome.

Here we outline some initial ideas towards an approach where nodes themselves autonomously locate, select, install and assess protocols to provide services to the application level without user intervention. This clears the way for rapid deployment of services without relying on user intervention. Essentially what we are proposing is a way to automate the social bootstrapping process.

## 2. AUTOMATIC SOCIAL BOOTSTRAPPING

In the same way that manual operator boot procedures of the past were replaced with automated sequences (basically simulating what an operator used to do), so can we automate the process of social bootstrapping. To dispense with direct user intervention, however, we need to replace their informal methods for the location of new packages and their judgement concerning how to evaluate them. Additionally, we wish to maintain the social utility of deployed protocols such that, in general, nodes will select protocols that increase social welfare rather than purely individual needs.

We also require a minimal and general approach that could be applied over many domains rather than being tied to implementation or application specifics. Furthermore, we need to provide a mechanism whereby new protocols can be easily introduced into the system such that they automatically spread yet exclude the possibility of malicious viruses utilizing the mechanism.

Since currently these tasks fall on the shoulders of human users, can we capture them in some algorithmic way? In or-

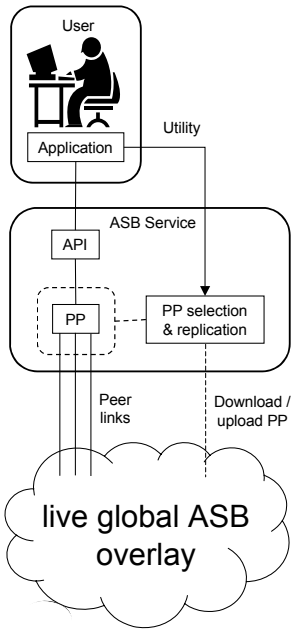


**Figure 1: Schematic of a single node from the user down to the ASB overlay. The protocol package (PP) contains both protocol code (implementing the API that applications use) and the current view (links to neighbor nodes). Lines indicate bidirectional information flow, arrows indicate directed flow.**

der to do this, we need some simple kind of "theory" of social behavior which can be easily encoded into a protocol. We argue that a simple rule based on models developed within evolutionary game theory [11] can be applied in various application contexts giving good results. The essence of the rule is trivially simple: if some other peer node is doing better than me, then I should copy its protocol and peer links. What is interesting is that from such a simple rule complex dynamics result which favor the spread of socially beneficial protocols and resist invasion by socially destructive ones, even when these may benefit individual node interests. In previous works, we have demonstrated how variations on this simple rule can produce protocols that self-organise socially cooperative networks [6, 7, 8]. In these works, however, protocol variants emerged through randomized mutations producing dynamicity of the networks. Here, as will be seen, we propose the idea of *Automatic Social Bootstrapping* (ASB) for stable networks in which protocols are explicitly introduced into the network.

## 2.1 The ASB Service Protocol

We concentrate on distributed systems in which nodes form a P2P overlay network. We assume nodes maintain a bounded view into the overlay comprising a set of links to other nodes. We assume each link is symmetric which means that the overlay can be represented as an undirected graph[1]. In addition, we assume nodes run user applications

[1]This assumption is carried over from the "tag" mechanism on which the protocol is based [6]. In on-going work we are experimenting with relaxing this assumption.

```
Periodically each node:

i ← this node
j ← SelectRandomNode()                  // select a random node j
if U_j » U_i then                       // utility of j higher?
        PP.protocol_i ← PP.protocol_j   // copy protocol of j to i add j
        PP.view_i ← PP.view_j ∪ j       // copy view of j to i add j
```

**Figure 2: The basic ASB service selection protocol. Nodes periodically compare their utility with randomly selected others. If the utility (U) is significantly higher, then the protocol package (PP) is copied, overwriting the existing PP. In addition the view stored in the source PP is copied and a link to the source node is added.**

which draw on services offered by P2P protocols. Protocols are implemented as code which maps an API to underlying requests passed to nodes within the views. Additionally, protocols may service requests from other nodes within their view. Such requests may require action at the application level.

We define a Protocol Package (PP) as an instance of a protocol plus an associated view. PP's are selected and replicated between nodes based on a utility produced by the application. The utility should be some numerical value that allows comparison between nodes using the same API. It should be periodically recalculated by the application to reflect the current quality of service given by the PP. Figure 1 shows a schematic diagram of a single node using a single application which draws on services supported by a PP.

The ASB service is itself a P2P protocol (a meta-protocol) that selects and replicates other protocols (by copying the PP) from the network. The ASB service protocol itself is not replicated or selected from the network — it is fixed and would need to be installed in a conventional manner or be supplied as part of an operating system. The ASB service protocol applies a utility-based replication approach as shown in Figure 2. When a PP is replicated, the view is amended by adding the source peer itself. If this causes a view to exceed its maximum size, then a randomly selected old link is removed from the view to make space for the new link. The ASB assumes a random sampling service over all peers within the ASB overlay. This is not implemented by ASB itself. A random sampling service could be provided using existing distributed P2P methods such as the Newscast protocol [9].

## 2.2 Deploying New Protocols with ASB

Given an existing ASB overlay, deployment of a new protocol follows three stages: (a) seeding; (b) injection; (c) deployment. Stages (b) and (c) are automated through the ASB service. Stage (a) requires programmer intervention. Assuming the ASB service is open (as we do), then anyone may generate a new protocol that implements the API. Once produced, the protocol needs to be "seeded" into the ASB overlay. In order to do this, a minimal ASB overlay which executes the protocol needs to be run live. This takes the

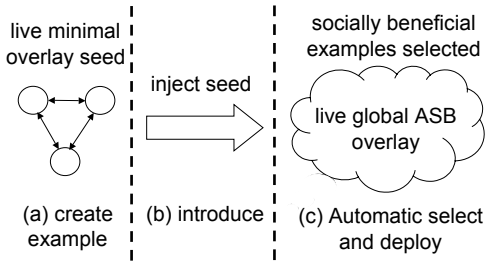


**Figure 3: The three stages of new protocol deployment in ASB.**

form of an "example" of a functioning protocol.

Once an example seed is live, it can be injected into the global ASB overlay. This involves wiring a single node from the example seed to a single node in the global overlay. Then, automatically, the new protocol will be replicated into the ASB overlay if nodes within the overlay decide to copy it based on the utility comparisons. This process occurs without the necessity of user intervention or awareness. Figure 3 illustrates the stages in the deployment process.

In this approach, the large-scale testing of the example seed is via the ASB overlay itself — if it is functioning properly it will only select and replicate protocols that are of social benefit. We envisage this as a final step in testing of a new protocol: rather than having to engage in large-scale testing, a developer can simply inject the example seed and observe. If the protocol is not selected then figure out why, modify and re-inject. This is akin to a live edit-compile-debug software development cycle.

# 3. SIMULATION RESULTS

Using simulations, we tested our ASB approach with a minimal canonical application. We had nodes periodically play the single-round, two-player Prisoner's Dilemma (PD) game with their neighbors (nodes in their view). Utility was set to be the average of individual payoffs received from playing different instances of the game.

We chose this game because it is a stress test for the conflict between "social benefit" and "individual rationality" — in the single-round PD game, there is always an individual temptation to act in an anti-social way because this will improve individual performance [2].

## 3.1 The Prisoner's Dilemma

In the two-player, single-round Prisoners Dilemma (PD) game, two players interact by selecting one of two choices: to "cooperate" (C) or to "defect" (D). For the four possible outcomes of the game, players receive specified payoffs. Both players receive a reward payoff ($R$) and a punishment payoff ($P$) for mutual cooperation and defection, respectively. However, when individuals select different moves, different payoffs of temptation ($T$) and sucker ($S$) are awarded to the defector and the cooperator, respectively (see Table 1). Assuming that neither player can know in advance which move the other will make and wishes to maximize her own payoff, the dilemma arises through 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. No player wants $S$ because its the lowest payoff. No matter what the other player does, by selecting D, a player always gets a higher score than it would have obtained if it had selected C. D is therefore the dominant strategy hence an ideally rational player would always choose D.

| | C | D |
|---|---|---|
| C | $R, R$ | $S, T$ |
| D | $T, S$ | $P, P$ |

**Table 1: The Prisoner's Dilemma payoff matrix.**

Therefore, the dilemma is that if both players select a cooperative (C) move, they are jointly better off (receiving $R$ each) than if they both select D; but selfish players will select mutual defection, receiving only $P$ each, because of the individual incentive to select defection. We select this game as a minimal test that captures a range of possible application tasks in which nodes need to establish cooperation and trust with their neighbors but without central authority or external mechanisms that enforce it.

## 3.2 Simulation Details

Application level behavior involves each node playing the PD with randomly selected neighbor nodes. We implement an API with a single method $PlayPD()$ which "plays" a single round of PD with a randomly chosen neighbor and returns the resulting payoff to be used for the utility.

We evaluated the ASB service by testing it with two simple PP protocol variants: 1) the D-protocol which always plays D; 2) the C-protocol which always plays C. We initialized the simulated ASB network with all nodes opting for the D-protocol with random views (a random graph). Each node had a maximum view size of 20 neighbor links. We set the PD payoffs [2] to $T = 2$, $R = 1$, $P = 0$, $S = -1$. For utility comparisons, we used a difference threshold of 0.5 to be considered significant. In other words, the ASB service copies the PP from another node if its utility is at least 0.5 greater.

For simulation purposes, nodes were selected to play the PD game randomly from the population. If selected, a node makes one $PlayPD()$ call. This causes a random neighbor to be selected and a single round of PD to be played between the pair. The PD move selected by each node (C or D) is dictated by the protocol the node currently holds within its PP. Payoff values are then returned by $PlayPD()$ to the application as dictated by the PD payoff matrix (see Table 1). The running average of these payoffs is stored by the application as the utility.

In each cycle, $10N$ nodes were selected to play the PD, where $N$ is the total population size. This means that on the average, each node initiated 10 PD games per cycle. At the end of each cycle, after all the games had been played, $N/2$ nodes were selected, in-turn, randomly from the population to execute the ASB protocol. When a node executed the ASB protocol, it selected a random node from the population, compared utilities and possibly replicated a PP as previously discussed (see Figure 2). After $N/2$ nodes had executed the ASB protocol, the next cycle commenced and

[2]Strictly T should be less than 2 to satisfy the PD constraints. We obtained similar results with T slightly less than 2.

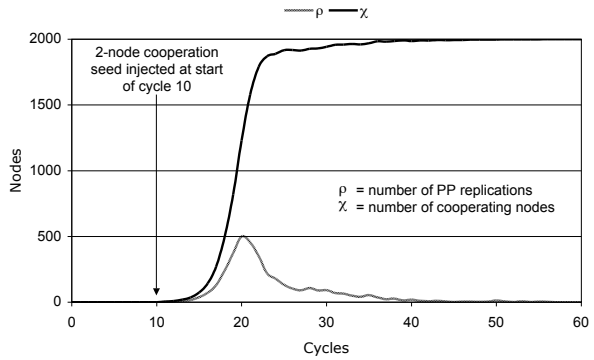


**Figure 4: A typical simulation run for a 2000-node network. All nodes are initialized to the D-protocol and linked as a random graph topology. The $\chi$ curve shows the number of nodes executing the C-protocol. The $\rho$ curve shows the number of PP replications performed by the ASB service at each cycle. At cycle 10, an "example seed" of the C-protocol is injected into the population. The ASB service quickly replicates the protocol through the entire network.**

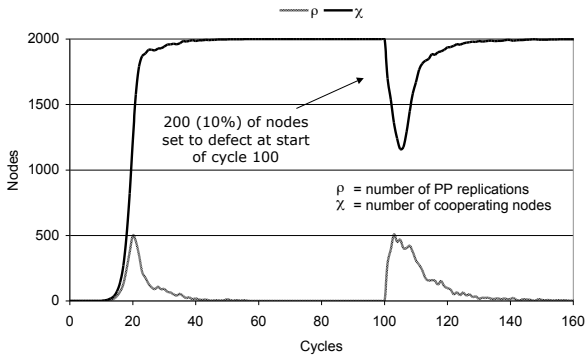


**Figure 5: A simulation run with the same initial conditions as those of Figure 4. To test robustness, 10% of the nodes are set to the D-protocol at cycle 100. After an initial spreading of the D-protocol, the system rapidly recovers to a full C-procotol population.**

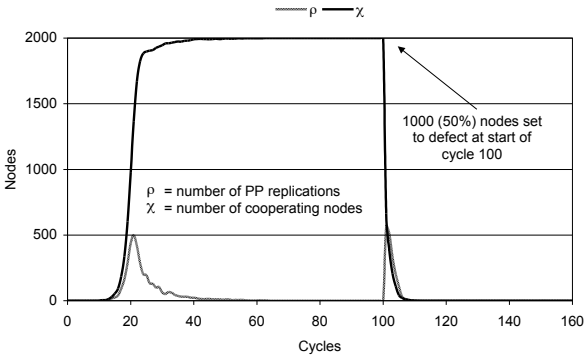


**Figure 6: A simulation run with the same initial conditions as those of Figure 4. When 50% of the nodes are set to the D-protocol at cycle 100, the system degrades back to a population of D-protocols, never to recover again. Hence, under these extreme conditions, the ASB service fails.**

PD games were played again. This process was repeated until some maximum cycle was reached.

Although our simulation separates game playing and replication into two distinct phases (which aids analysis), the randomized ordering of the node selections allows us to be confident that this separation does not introduce artifacts that would not be present in a system where these actions were interleaved asynchronously.

## 3.3 Typical Runs

Here we give three typical runs for populations of 2000 nodes. We began each simulation with all nodes containing a PP storing the D-protocol and links to randomly selected nodes (a random graph). In each case, the population was static with all nodes executing the D-protocol. This means that the average node utility is $P$.

Then an "example seed" of the C-protocol was created by selecting two nodes from the population, setting their protocol to the C-protocol, removing all of their existing links and creating a single link between them. This "example seed" was then "injected" into the population, at cycle 10, by making a link between one of the example nodes and a randomly selected node from the population.

As can be seen from Figure 4, very quickly (after cycle 10), the C-protocol spread quickly throughout the population leading to a stable population in which all nodes stored the C-protocol. This increases the overall utility by increasing the average node utility from $P$ to almost $R$. This means that the socially beneficial protocol was selected by the ASB service.

We checked the robustness of the C-protocol populations by injecting various proportions of D-protocol example nodes back into the population after the C-protocol had dominated and stabilized in the population. We found that in our 2000 node population we needed to set 50% of the population to the D-protocol in order to cause the population to fallback into the less socially beneficial situation in which the D-protocol establishes itself on all nodes in the population. Figure 5 shows the population recovering from the injection of 10% D-protocol nodes at cycle 100. Figure 6 shows the population falling back to the D-protocol after 50% D-protocol nodes were injected at cycle 100.

## 3.4 Interpretation of Results

These very initial results do not prove our method in general. They show that for a very simple scenario, in which we employ a canonical test case (the PD), the ASB approach appears to operate in a desirable manner. It is a kind of "social ratchet" selecting the socially beneficial C-protocol from a single example yet resisting the socially less beneficial D-protocol even when large proportions of nodes are set to it. We found that the system finally failed when more than 50% of nodes were simultaneously set to the D-protocol. However, it is important to realize that only a single two-node

example of the C-protocol seed would be sufficient to flip the system back into the socially beneficial C-protocol.

Further experiments (not shown here) indicate that these results are scalable and robust to node failures but more experimentation needs to be done to fully verify this. Also, although it appears that similar results are obtained with different PD payoffs, we also find that certain payoff values cause the ASB to fail. Again on-going work is looking into this.

Why does ASB select the socially beneficial C-protocol when individual nodes would seem to do better with the D-protocol? We have dicussed this counter intuitive result elsewhere at length [6, 7]. It involves a topology-based process in which the overlay isolates D-protocol nodes so they end-up only connected to other D-protocol nodes. At that point, D-protocol nodes can gain higher utility by copying C-protocol nodes. We do not discuss this here in detail.

# 4. CONCLUSIONS

We have proposed the notion of automatic social bootstrapping (ASB) and the problems it might address in distributed peer-to-peer systems. ASB is essentially a meta-protocol that selects other protocols that are socially beneficial — good for the network as a whole. ASB is both open and completely distributed, requiring no central authorities, information storage or control, yet aims to attain a socially beneficial outcome even in the presence of egotistical peers who only care about their own utility. We outlined a sketch of how such a system might operate and presented some high-level initial simulation results applied to a simple scenario using the canonical Prisoner's Dilemma game.

The ideas we have presented are currently at an early stage and raise many questions that need to be addressed. For example, it is currently unclear exactly how a utility can be meaningfully defined and compared between nodes, especially if they are different applications or application usages. Also, major security questions arise when allowing for automatic code replication and execution. For example, how can any ASB type protocol be certain to only select non-malicious protocols? Furthermore the proposed protocol would appear vulnerable to sybil attacks [4] and possibly whitewashing attacks where nodes change identity regularly. However, an increasing number of solutions are being proposed to address these problems [5]. In addition, recent experiments with a protocol similar to ASB produced results that indicate some robustness to possible attacks [1].

Can the results from the simple example of the PD scale to more realistic P2P protocols? This is an open issue. So far we have shown how similar protocols to ASB can be applied to file sharing [6] and job sharing relying on specialization [8]. But we currently have no general predictive theory for what kinds of tasks can and can not be tackled in this way so we have relied on incremental simulation models.

Dynamic protocol platforms have been developed that allow for rapid run-time protocol adaptation without disturbing running applications [10]. These kinds of platforms could provide the essential packet-level infrastructure on which meta-protocols such as ASB would execute.

A substantial amount of research effort is required even to assess the feasibility of an ASB approach but we believe it is not an unreasonable direction for further work.


## Acknowledgements

We would like to thank Mark Jelasity, Simon Patarin, Sefano Arteconi, Giovanni Rossi, Andrea Marcozzi and Edoardo Mollona, all from the Department of Computer Science at the University of Bologna, for many discussions relating to aspects that influenced the paper.


# 5. REFERENCES


[1] Arteconi, S. and Hales, D. *Greedy Cheating Liars and the Fools Who Believe Them.* University of Bologna, Dept. of Computer Science, Technical Report UBLCS-2005-21, (available at: http://www.cs.unibo.it/pub/TR/UBLCS/2005/). 2005.

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

[3] Cohen, B. *Incentives Build Robustness in BitTorrent.* Presented at the 1st Workshop on the Economics of Peer-2-Peer Systems, June 5-6, Berkley, CA, (available at: http://www.sims.berkeley.edu/research/conferences/p2pecon/) 2003.

[4] Douceur, J. *The sybil attack.* Proc. of the IPTPS02 Workshop, Cambridge, MA (USA), March 2002, (available at: citeseer.ist.psu.edu/douceur02sybil.html), 2002.

[5] Feldman, M. et al. *Robust Incentive Techniques for Peer-to-Peer Networks.* Proc. 5th ACM Conf. Electronic Commerce (EC 04), ACM Press, pp. 102-111. 2004.

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

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

[8] Hales, D. *Choose Your Tribe! - Evolution at the Next Level in a Peer-to-Peer Network.* In Engineering Self-Organising Systems. Proc. of the 3rd Workshop on Engineering Self-Organising Applications, LNCS 3910, Springer. 2006.

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

[10] Patarin, S. and Makpangou, M. *Pandora: an efficient platform for the construction of autonomic applications.* In Self-Star Properties in Complex Information Systems, LNCS 3460, Springer.

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

[12] Strahilevitz, L. *Charismatic Code, Social Norms, and the Emergence of Cooperation on the File-Swapping Networks.* Virginia Law Review, Vol. 89, (available at: http://ssrn.com/abstract=329700), 2003.

[13] BitTorrent Webpage: http://www.bittorrent.com

[14] Skype Webpage: http://www.skype.com

[15] Edonkey Webpage: http://www.edonkey.com