Deliverable 5.3.2
Applications of bio- and socio-inspired algorithms in information systems

Due date of deliverable:	December 2007
Actual submission date:	December 2007
Dissemination level:	PU – public
Work Package 5.3:	Biologically and Socially Inspired Design for Dynamic Solution Spaces
Participants:	UniBO RAL UPF Telenor

Authors of deliverable:
Stefano Arteconi (arteconi@cs.unibo.it)
Toni Binci (bincit@cs.unibo.it)
Ozalp Babaoglu (babaoglu@cs.unibo.it)
David Hales (hales@cs.unibo.it)
Alberto Montresor (montresor@dit.unitn.it)
Mark Jelasity (jelasity@inf.u-szeged.hu)
Gian Paolo Jesi (jesi@cs.unibo.it)
Fabio Picconi (picconi@cs.unibo.it)

Abstract

This report comprises the complete D5.3.2 deliverable as specified for workpackage WP5.3 in Subproject SP5 of the DELIS (Dynamically Evolving Large-scale Information Systems) Integrated Project.

The essential goal of the DELIS project is to understand, predict, engineer and control large evolving information systems. The main aim of this workpackage is to understand how algorithms originating within biological and social sciences can be applied to applications in distributed computer systems. It is not our aim to present finished applications but to attempt to gain some generic insights that might inform a method by which other such algorithms can be modified.

To this end we have shifted focus from our planned work from the previous implementation plan somewhat. This shift has been motivated by two factors: firstly, we became aware (with some horror) that in realistic environments all our previous approaches were highly vulnerable to malicious attacks, not at the application level in which free-riding and selfishness were somewhat controlled, but in the underlying peer-to-peer gossiping infrastructure. All the previous applications we have considered (including broadcast and content replication) relied on a self-organised gossiping infrastructure. But this is highly vulnerable to so-called "hub" attacks. Hence we decided to focus on applying our group selection approach to this infrastructure itself. Many existing proposed, and some deployed, systems utilise gossip based approaches and our proposed defence mechanism is backward compatible with such systems.

Secondly, we realised that, it would be productive to apply our approach to existing and highly deployed systems. To this end we have focused much effort on the existing BitTorrent system. Specifically we were looking for socially inspired approaches that could be easily retrofitted to this protocol to defeat existing (and potential future) selfish, exploitative or unfair clients. Again our proposed approach is backward compatible with existing clients and in theory could be deployed and empirical results gained from interactions "in the wild" but this is on-going work.

Finally we report on work which applies a biologically inspired model of firefly flash synchronisation to produce self-organised and robust synchronisation in peer-to-peer overlay networks.[1].

[1] Most papers produced within DELIS are available from the DELIS website as DELIS Technical Reports. Where this is the case references are appended with the DELIS Tech Report number in square brackets. This indicates the paper was produced within the DELIS project, not some other project.

Contents

1 A secure gossip-based peer sampling service

Here we overview recent work reported in [8]. P2P systems, without central servers, need to provide some method of initiating and maintaining connections between the nodes that comprise them such that all nodes form a single component. A partitioned network reduces the efficiency of the system, for many tasks, because nodes in different components cannot communicate. This can be a significant problem in unstructured systems which operate under highly dynamic environments with nodes constantly entering and leaving the system.

One general approach to this problem is to implement a protocol that maintains a connected overlay network between nodes which approximates a random topology. An overlay network topology consists of each node maintaining logical links to other nodes. A logical link consists of a node identifier that is sufficient to establish communication using some underlying network infrastructure (e.g. and IP address and port number over the Internet).

It is well known that a random network topology can maintain a fully connected network that is highly robust to benign node and link failure. Additionally, a random network offers short paths between any two nodes in the network which is valuable for many kinds of P2P tasks (e.g. broadcasting or routing messages between nodes).

A specific method for maintaining robust overlays with random-like topologies is through gossiping. Gossiping protocols rely on the randomized spreading of information between neighbors in a network. Typically, nodes maintain a set of neighbor links (a so-called cache or view) which indicates their currently neighbors. Periodically, each node selects some random neighbor from its view and communicates some information – i.e. gossips – which the receiving node may store and later forward to its own neighbors. The approach is loosely analogous to individuals in a social network gossiping between themselves or the spread of an epidemic in a population.

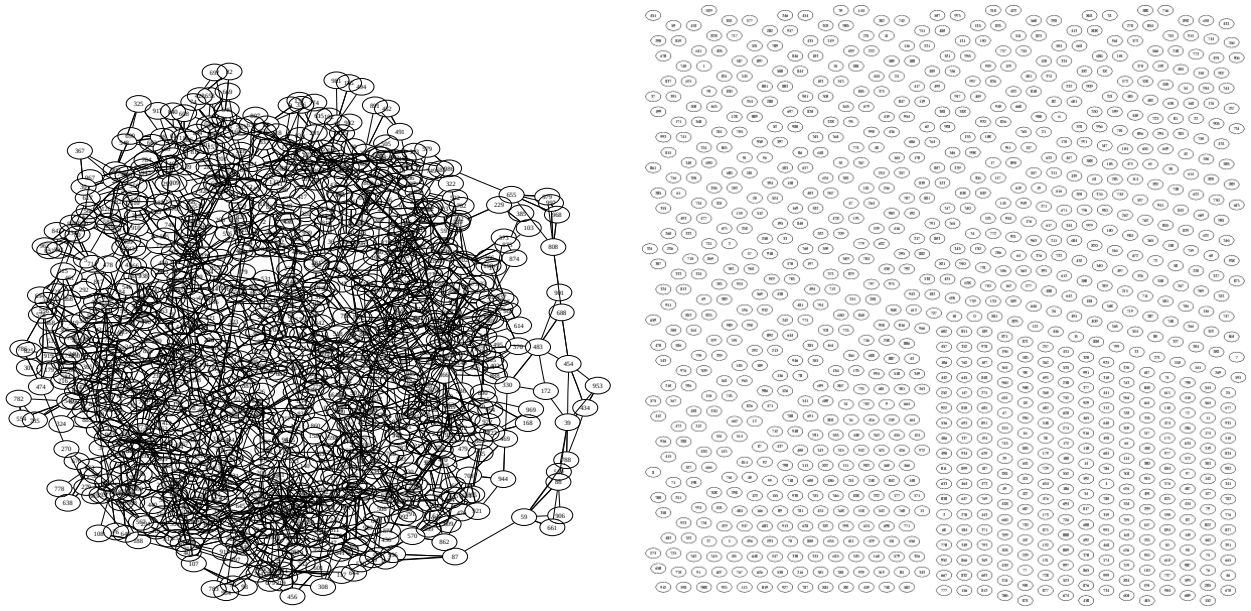
Gossip approaches are attractive because they spread information quickly and robustly over networks yet require only simple protocol implementations. A number of gossip-based protocols exist to maintain random overlay networks in unstructured P2P systems [7, 13]. Although implementations vary, the basic mechanism involves nodes gossiping their current neighbor links. In this way, using suitable update functions in the nodes, the cache (or view) can be kept up-to-date and maintain a random-like connected topology under conditions of high dynamism – where nodes constantly join and leave the network.

Ironically, however, the power of the gossip approach to spread information quickly over the entire network can become an achilles' heel if it is exploited by malicious nodes who wish to defeat the system by spreading false information to partition the network. Because information spreads so quickly in gossip networks the problem for non-malicious nodes is that *by the time they identify other nodes as malicious it is too late for them to take action* since they are already disconnected from the network - i.e. their view is completely polluted by the malicious nodes.

We show that by maintaining multiple views over the network (multiple neighbor lists) nodes can identify malicious nodes before it is too late such that they can take appropriate action by placing them on an individually maintained black-list. The approach is adaptive, allowing formally blacklisted nodes to be white-listed if their behavior changes and vice versa. In addition no reputation information must be shared between nodes because blacklists and white-lists are only individually maintained. In other words, we present a fully decenralized Secure Peer Sampling Service (SPSS).

1.1 The problem - the hub attack scenario

Here we describe an example of an attack scenario applicable to a gossiping topology maintenance approach. This attack is a form of the so-called "hub attack" (see [9]) and its effect is summarized in Figure 1. In a hub attack malicious nodes attempt to get other nodes to connect exclusively to them. If this is achieved, then the malicious nodes can exit the network leaving their former



(a) A healthy random graph topology

(b) The graph after malicious attack

Figure 1: Overlay topology before and after a hub attack: the random graph depicted in (a) becomes fully disconnected in (b). The graph out-degree (constant) is set to 20, but only 3 links per node are displayed for clarity. Less than 20 gossiping cycles are required to disrupt the graph. Network size is 1000 nodes.

neighbors without any valid neighbors and hence partitioning them from the network. To illustrate the problem, consider a gossiping network in which each node maintains a list of c neighbors ($c = 20$), called its *cache* or *view*. When the network is performing correctly the peers are wired in a random graph topology with out-degree c and the graph is continuously rewired over time. The elements of these caches are continuously updated and exchanged between nodes during gossiping interactions.

At some point, k nodes in the system (e.g., $k = c$ in the example) start colluding and behaving maliciously exchanging forged caches; then, after a short amount of time spent in their malicious activities, they leave the network. The forged caches exchanged by the attacker nodes hold the identifiers of the other malicious nodes; this exact malicious content of the cache is always replayed at every gossip exchange.

Figure 1(a) shows the overlay in normal conditions; Figure 1(b) shows the same graph after the malicious nodes exit: within a very short time the original overlay is completely disrupted. The infection rate proceeds quickly, as can be expected from a gossiping protocol. In fact, every well-behaving node will accept the attacker's cache with high probability (see Section 1.2); this means that in just one exchange a node will have all its cache polluted by malicious identifiers if it is directly connected to a malicious node. In other words, all its neighbors are now malicious nodes and it can no longer *initiate* a gossip exchange with a non-malcious node.

Only a nondefeated, nonmalicious, node B can help a defeated one, say A, provided B has As identifier in its local cache. However, when B contacts A, we can expect that after the exchange half of the local caches of A and B, respectively, will be polluted with identifiers of malicious nodes. As a consequence, both nodes will have 50% of their cache polluted; therefore, even contacting a nonmalicious node will generally also spread the pollution and increase the chance that a nondefeated node contacts a malicious one.

There is no way for a nonmalicious node to identify a malicious one as they seem to play fairly; however, as the attackers always pass on the same cache (or a very similar one), it is easy for any

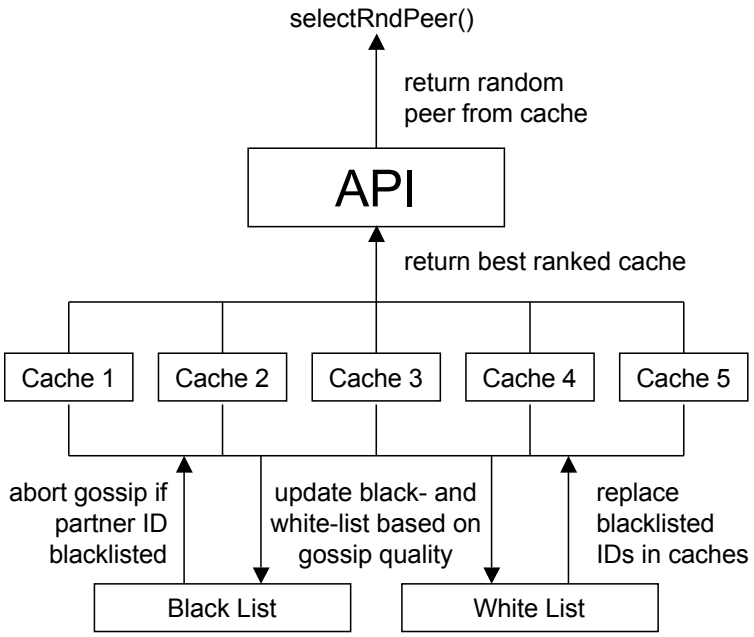


Figure 2: Schematic of the decentralized SPSS; it maintains multiple caches to support multiple random overlays. Black and white-lists screen incoming gossip requests and refresh malicious cache entries.

node to keep track of the last cache provided by a neighbor in order to detect them. Sadly, when a non-malicious node detects the bogus cache replayed by the same neighbor, *it is too late to react* since the node cache is completely filled with malicious identifiers. When each node's cache is completely polluted, the malicious nodes may decide to leave the network, leaving it in a completely disrupted state without any hope of recovery, as shown in Figure 1 (b).

In this section we have introduced the general idea of a hub attack as applied to a gossip based topology manager service. In the next section we specify in a little more detail the generic gossip protocol approach and the specific protocol variant we used for the purposes of our simulations.

1.2 The solution - group selection inspired secure peer sampling service

Here we outline our distributed secure peer sampling service (SPSS) that can deal with the hub attack just described.

Multiple overlays: As we have seen the main obstacle to prevent and detect the hub attack is represented by its *high spreading speed*. Such a high speed leaves no time for the peers to make any successful guess about the identity of the attackers.

The basic idea for the fully distributed SPSS is based on using multiple overlay graphs combined with black and white lists. By maintaining multiple overlays (each node having a set of distinct instances of views) means the neighborhood at every instance will be distinct with very high probability because the overlays have independent random-like topologies. Essentially, the multiple view caches over the same node population, which every node adopts, give each peer a snapshot of what is going on in distinct (random) neighborhoods of the overlay. We call *extra caches* the set of view caches belonging to each peer; every cache in the set is a random snapshot of a distinct overlay

We assume an attack model in which a set of k colluding attackers attempt to pollute the caches of all nodes.

Each node can monitor the pollution ratio by looking at its extra caches. Since the network

population of all the overlay instances is the same, all the extra caches will become polluted by the same k malicious node IDs, if no checking action is performed. Due to the random nature of the available overlays, it is very unlikely that an attacker could defeat all caches of the same victim peer in a short time window. Essentially, the multiple caches are useful in order to perceive how malicious nodes are spreading the infection from distinct directions over distinct overlays. Due to the spreading infection, we expect that common node ID patterns will emerge in all (or in the majority) of the caches.

Quality rating: Each peer can build a set of statistics in order to guess or detect who are the malicious nodes from the emerging patterns. This knowledge base is stored as private, local black- and white-lists that it is never exchanged among neighbors. This obviates the second-order issue of malicious nodes spreading incorrect reputation information.

During a gossip exchange, both parties *rate the quality of the exchange*. The quality rate is given by the number of items lying in the intersection of the exchanged caches among node A and B: $r = |\{cache_A \cap cache_B\}|$. This quality rate influences the probability to reject the gossip exchange with this current neighbor. Essentially, when two caches are similar (or identical) it is likely that the current neighbor is a malicious node and with high probability it should not be accepted. The probability to abort the exchange is proportional to the fraction of the common IDs found among the two caches: r/c, where c is the usual cache size.

The rank results are collected in the node's *knowledge base* comprising a blacklist and whitelist. The information collected in this structure is refreshed according to an aging policy to avoid that any wrong guess would have unbounded consequences over time.

Any attempt to exchange views with a blacklisted neighbour is declined. In addition, when a node suspects one of its caches is polluted, it tries to refresh the cache randomness by substituting the currently blacklisted node IDs with high quality (whitelisted) node IDs collected during the previous exchanges.

During the protocols execution, one or more cache can be defeated by the attackers. However this is not critical, as the cache will be restored as soon as the node has collected a suitable knowledge base. It is very unlikely that all node's caches become polluted in a short amount of time; in this unlucky condition and if the knowledge base is not ready or not correct, the only chance for a node is to be contacted by a well behaving node in order to partially restore at least one of its caches. Figure 2 shows a schematic of the main components maintained by the protocol within each node.

The algorithm: Our distributed approach is focused on the knowledge base each node has to build. Essentially, the knowledge base is represented by two list structures: blacklist and whitelist; the former holds high frequency and low quality rated node IDs, while the latter holds high quality rated node IDs. We do not set any explicit size limit for these structures and, as a consequence, their size may grow to the actual network size. However, due to presence of an aging policy, their actual size is much less than the theoretical maximum. The SPSS algorithm pseudo-code executed by a node A is the following:

1. Select a random neighbor $B \notin$ blacklist
2. Compute the rank value r with B; with Prob(r/c) decline and blacklist B, otherwise accept the gossip exchange, and whitelist B

These steps are performed in each cycle for every available cache. Two additional actions are performed concurrently by two threads at the end of each cycle. The first action is to purge the blacklist and whitelist according to an aging policy; the second action instead, is to repopulate the caches suspected of being polluted (if any): each node ID in the cache listed in the blacklist is substituted by a random node ID picked from the whitelist.

Another issue is to clarify how node IDs can be inserted and swapped from the blacklist to the whitelist and vice-versa. When a node ID has to be inserted in the blacklist for the first time, a standard TTL value (2 cycles) is bound to the stored ID; if the ID is already present instead, its TTL value is reinforced (i.e., doubling the current TTL value).

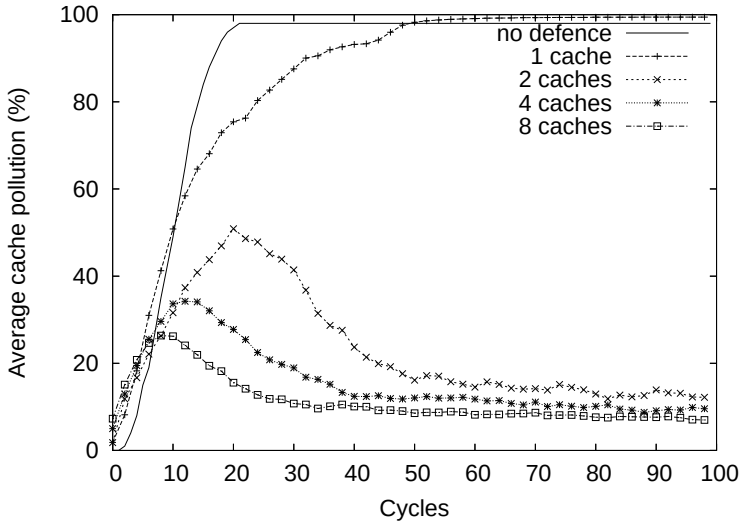


Figure 3: Fully decentralized SPSS algorithm. The average pollution level in the caches is shown over time; multiple distinct caches per node are compared (i.e., 1, 2, 4 and 8 caches) for a network of size 1,000 nodes. 20 malicious nodes are involved in the attack.

Why it works: It is important to note that having multiple caches belonging to distinct overlay instances is very different from having a single overlay with a large size. Multiple caches add extra randomness to the node's state and avoid to be defeated in just one gossip exchange; in addition, in extreme conditions (i.e., when the set of attackers is larger than the cache size) they stil give the chance to identify the attackers.

The value added by multiple PSS overlays is that the infection proceeds from distinct multiple paths. These dynamics gives each peer more time to detect the most frequent node IDs that appear in their caches.

A higher-level protocol or application working on top of this fully distributed SPSS can see just a single cache. This maintains a seamless integration with the standard selectRndPeer() API function. A smart implementation of the fully distributed SPSS can dynamically export the *current best cache* according to concentration of suspected malicious nodes currently listed in the knowledge base (see Figure 2).

Group selection: The multiple caching concept originates from previous socially inspired evolutionary models of "group selection" [6, 5]. In these models anti-social behavior between nodes was avoided by allowing nodes to form and move between different clusters or groups in the population based on utility value comparisons. Essentially, nodes evaluated the quality of their neighbors by measuring the effectiveness of interaction with them over time – involving some application level task – and represented this as a utility value. By comparing utilities with other randomly selected nodes and copying the neighborhoods (caches) of those with higher utility, nodes could avoid interaction with anti-social free-riding nodes. In this approach nodes maintained a single overlay and made intra-overlay movements to find better (higher utility) neighborhoods.

For the distributed SPSS we implemented a similar scheme by allowing each node to store multiple caches and only selecting the best cache based on a measure of utility expressed as cache quality. From the point of view of what is passed to the API, nodes are constantly shifting between different views of the network since each cache represents a different set of neighbors. Furthermore, when the quality for a particular cache becomes low due to possible identification of malicious information, it is wiped and reinitialized from the whitelist, hence low quality caches are dropped.

Hence in SPSS nodes do not move between distinct groups or clusters in a single overlay but maintain and effectively move between distinct overlays (inter-overlay movement) comprising the same population of nodes but in different topological configurations. Hence what is being selected here by each node is the overlay which produces the best cache quality at each given point in time. Since all nodes actually stay in all overlays at all times (by maintaining a fixed number of multiple caches) this approach is less a form of evolution and more a form of redundancy with dynamic selection.

1.3 Summary

In this section we have given an overview of the main ideas and results from [8]. We have presented a gossip based secure peer sampling service (SPSS) which is inspired by a kind of group selection process, where the groups are multiple gossip overlays. This is essentially a decentralised version of [9] with nodes selecting from experience which gossip overlay to use at any given time. This approach deals with the hub attack scenario discussed. However, there are of course other possible kinds of gossip based attacks that would not be stopped by the current SPSS - such as multiple identity or so-called "Sybil attacks" or if a high proportion of the population of nodes maliciously colluded. These could be topics for future work. The simulation work was performed on the Peersim platform [14].

2 An improved BitTorrent

In this section we discuss on-going work [11] concerned with improving the performance of the BitTorrent protocol. BitTorrent is a highly successful file-sharing system. The protocol is designed to co-ordinate peers so they can download files quickly from those who wish to share them by co-operatively helping each other to distributed chunks of the file. Peers working together to distribute are file are commonly known as a "swarm". This can be contrasted with traditional client / server approaches where all clients try to download from a single server.

In order to understand how BitTorrent clients may interact it is important to be clear about some terminology. BitTorrent attempts to produce an efficient solution to the sharing of a file among a set of peer clients. *Efficiency* in this context means utilising as much of the available upload bandwidth as possible under the assumption that peers generally have high download bandwidth. Upload bandwidth is considered as a scarce and perishable resource - if it's not used it's wasted.

BitTorrent incentivises uploading by using a form of tit-for-tat - essentially by disconnecting active links from other peers that do not upload. However it does not actively balance upload to download ratios between peers. Hence standard BitTorrent does not enforce fairness. *Fairness* in this context means peers will receive as much download data as they upload over time.

For BitTorrent to operate at least some number of peers are required to act altruistically by being the initial and on-going "seeders" of a file. Seeders posses the entire file and *altruistically* upload it to others without requiring anything in return. Incentives for altruism are not built into the protocol. It is simply assumed seeders must exist.

Given these aspects of the protocol peer clients may act selfishly either intentionally, by minimising their upload rate, or unintentionally, because of slow physical upload connections. In some sense it makes no difference to other nodes if a client is intentionally selfish or simply does not have the capacity to be any other way. The effect is the same – fast and co-operative peers will tend to be exploited by uploading far more than they download.

2.1 Selfish clients

Since the BitTorrent protocol is open this allows any programmer to modify the protocol and deploy a new client there are many such client variants available. Some are known to intentionally behave selfishly by attempting to exploit the unfairness that can be tolerated by the standard BitTorrent client.

Among the many selfish BitTorrent clients that exist "in the wild", two have recently caught the attention of the research community, most likely because they have been designed by academic researchers and their exploitation mechanisms are known.

The first is called BitThief [10], and was developed in the context of a research project at ETH Zurich. A BitThief client differs from standard BitTorrent in two ways: first, it connects to as many clients as possible within a swarm, rather than limiting the number of connections as BitTorrent does; second, BitThief never uploads any data to other nodes. Surprisingly, BitThief clients achieve download rates similar to standard BitTorrent clients, even though they do not contribute any resources to the network. The reason is that standard clients periodically upload some chunks of data to other nodes without asking anything in return. This mechanism, called optimistic unchoking, is normally used to bootstrap new nodes joining the swarm, but in this case is exploited by BitThief clients to get data from other nodes without ever giving anything back.

The second client is called BitTyrant [12], and was designed by the University of Washington at Seattle. BitTyrant exploits the fact that standard BitTorrent clients with fast upload connections will reciprocate (i.e., exchange data) with other nodes even if they download less data than what they upload to other nodes. The reason is that standard clients do not limit their upload bandwidth when they engage in a bidirectional data exchange with another, possibly slower node. BitTyrants strategy consists in looking for fast nodes, and exchanging data with them but uploading as little data as possible.

2.2 Enforcing fairness

In order to reduce exploitation by selfish or slow peers it is possible to enforce fairness by using a variant of tit-for-tat called *block level tit-for-tat* [2]. Essentially, peers disconnect from active connections that are exploiting them, by not providing a balanced upload / download ratio over time, and search for better connections. We created a simulation model of the BitTorrent protocol to test this. Figure 4 shows a comparison of download times for two sets of clients in a swarm and their associated average upload / download ratios for the standard BitTorrent protocol. Fast clients are assumed to have a 100kb upload rate and slow clients only 10kb upload rate. Notice that slow clients do as well as fast clients (or even better) at the expense of fast clients. A similar curve is seen when intentionally selfish clients are used. In this context peers are actually incentivsed to be selfish. Figure 5 shows what happens when block-level tit-for-tat is used. Slow peers take far longer to download the file. A similar curve is seen for selfish clients. Hence the download / upload ratios evidence a fairer outcome.

Here we show one further experiment in figure 6. Here fast and slow clients are separated into two independent swarms. As can be seen this further improves fairness and reduces slow clients download performance. Our on-going work involves allowing clients to actively move between swarms serving the same file in order to self-organise this situation. In order to ensure selfish and / or slow clients can not enter fair and / or fast swarms we are experimenting with a simple mechanism that allows nodes entering a new swarm to be assessed by existing members of the swarm and then only actively connected to them if they are fair clients.

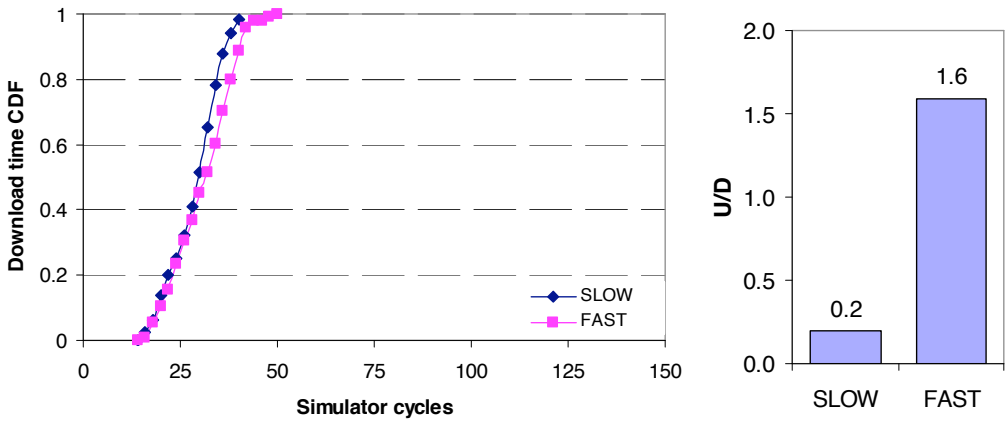


Figure 4: BitTorrent standard protocol – One swarm containing slow and fast nodes. The curves show Cumulative Download Frequency and the bar chart shows average fairness in terms of upload / download rates.

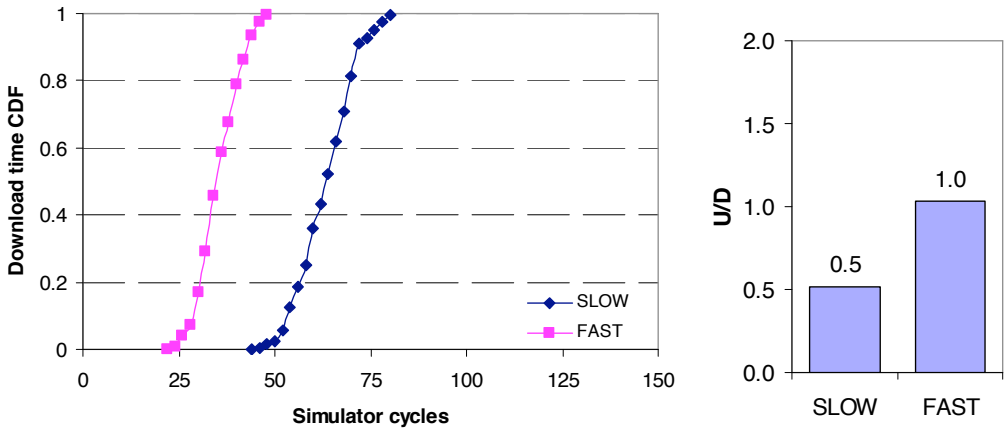


Figure 5: BitTorrent with Block-Level Tit-for-tat – One swarm with slow and fast nodes. The curves show Cumulative Download Frequency and the bar chart shows average fairness in terms of upload / download rates.

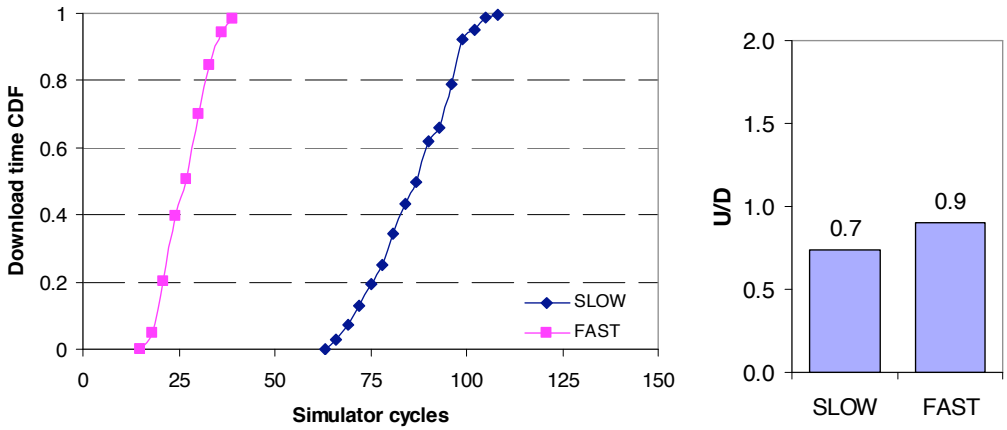


Figure 6: BitTorrent with Block-Level Tit-for-tat – Two swarms, one with slow nodes, other with fast. The curves show Cumulative Download Frequency and the bar chart shows average fairness in terms of upload / download rates.

2.3 Summary

We have outlined on-going experiments with a detailed simulation of the BitTorrent protocol that test mechanisms for enforcing fairness. The mechanisms discussed are backwards compatible with existing protocols and hence could be deployed and test "in the wild" by modifying client code and deploying. The approach should tackle exploitation by both intentionally selfish clients and slow clients that currently exploit fast and co-operative clients. The multi-swarm approach offers the potential to import a group-like selection process into BitTorrent where an ecology of swarms represent groups, with those producing fair outcomes for the clients within them being stable and growing whereas those containing exploiters will dissolve. We have not explored here the effect on efficiency that such fairness-based approaches may have. This depends on the ecology of existing clients and their capacities. It may be the case that some efficiency sacrifice is the price that has to paid to maintain fairness and disincentivize selfish client behaviour. This is on-going work which we hope to report on soon [11].

3 Firefly-inspired heartbeat synchronization in overlay networks

Heartbeat synchronisation strives to have nodes in a distributed system generate periodic, local "heartbeat" events approximately at the same time. Many useful distributed protocols rely on the existence of such heartbeats for driving their cycle-based execution. Yet, solving the problem in environments where nodes are unreliable and messages are subject to delays and failures is non-trivial. Here we briefly overview the work presented in [1].

In our protocol, nodes send flash messages to their neighbors when a local heartbeat triggers. They adjust their their next heartbeat based on incoming flash messages using an algorithm inspired by mathematical models of firefly synchronisation [3]. We report simulation results of the protocol in various realistic failure scenarios typical in overlay networks and show that synchronisation emerges even when messages can have significant delay subject to large jitter.

An innovative aspect of the Ermentrout [3] model is that it adjusts the local length of the flash cycle based on flashes received from neighbours, rather than adjusting purely the phase of the flash. This means that nodes need not initially have exact knowledge of cycle lengths but can converge to them. Previous synchronisation models have focused on adjusting the phase only.

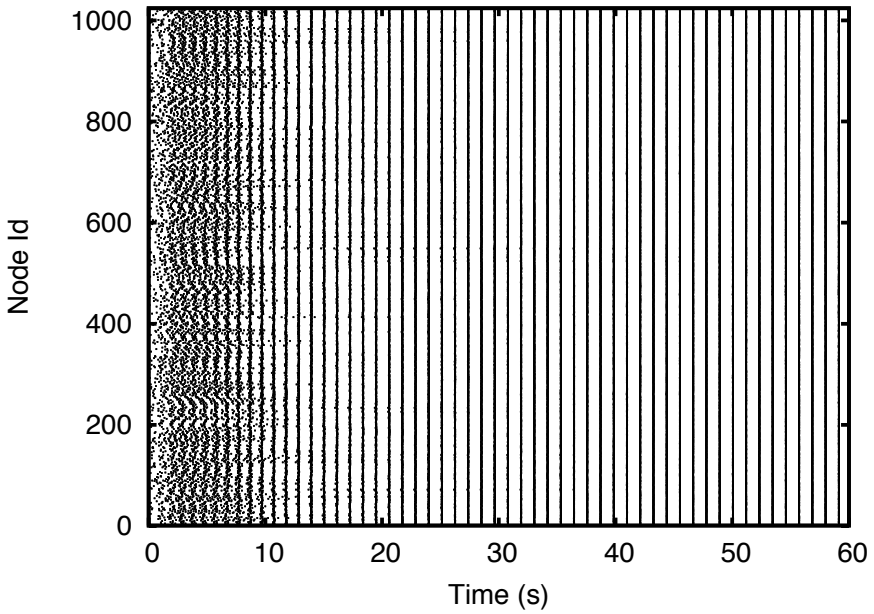


Figure 7: Flashes emitted by a network of 1000 nodes over an interval of 60 seconds. A black dot indicates a node flash. Initially nodes are unsycronized but quickly self-organise towards synchronous flashing.

Figure 7 shows results from a simulation run in which initially unsynchronised nodes quickly self-organise to synchronised flashing. The underlying network peer sampling service was provided by Newscast [7]. The simulations were implemented on the Peersim platform [14].

4 Conclusion

We have presented an overview of two application areas which broadly apply group-like selection dynamics to improve performance in the presence of malicious (in the gossip case with SPSS) and selfish (in the BitTorrent case) peers. We discuss further this group selection approach in the form of a design pattern applied to a number of application domains in [4]. We report on this work in DELIS deliverable D5.4.3 so we will not discuss this here. Suffice to say that the hub attack and BitTorrent applications we have presented here follow this general approach.

Open distributed systems, which are increasingly popular "in the wild" over the internet will always be victim to malicious and selfish behaviour. This is the price paid for the open nature of such systems. It is therefore increasingly necessary to develop broad design approaches and patterns that can be applied in different domains that can encourage, if not complete security or efficiency then, a disincentive for bad and anti-social behaviour.

References

[1] Babaoglu, O., Binci, T., Jelasity, M. and Montresor, A (2007) Firefly-inspired heartbeat synchronization in overlay networks. In Proceedings of the First International Conference on Self-Adaptive and Self-Organizining Systems (SASO2007), July 2007, Boston, MIT. IEEE Press. [DELIS-TR-0570]

[2] Bharambe, A., Herley, C. and Padmanabhan, V. (2005) Some Observations on BitTorrent. Proc. ACM SigMetrics 2005.

[3] Ermentrout, B. (1991) An adaptive model for synchrony in the firefly pteroptyx malaccae. Journal of Mathematical Biology, 29(6):571585, June 1991.

[4] Hales, D., Arteconi, S., Marcozzi, A., Chao, I. (2007) Towards a Group Selection Design Pattern. Technical Report UBLCS-2007-25, University of Bologna, Dept. of Computer Science. [DELIS-TR-0568]

[5] Hales, D. and Arteconi, S. (2006) SLACER: A Self-Organising Protocol for Coordination in P2P Networks. *IEEE Intelligent Systems* 21(2):29-35. [DELIS-TR-0370]

[6] Hales, D. and Edmonds, B. (2005) 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.* [DELIS-TR-0111]

[7] Jelasity, M., Kowalczyk, W. and van Steen, M. (2003) Newscast Computing *Technical Report IR-CS-006*, Vrije Universiteit Amsterdam, Department of Computer Science, November 2003, Available at: http://www.cs.vu.nl/globe/techreps.html#IR-CS-006.03

[8] Jesi, G. P, Hales, D., van Steen, M. (2007) Identifying Malicious Peers Before Its Too Late: A Decentralized Secure Peer Sampling Service. Proceedings of the First International Conference on Self-Adaptive and Self-Organizining Systems (SASO2007), July 2007, Boston, MIT. IEEE Press. [DELIS-TR-531]

[9] Jesi, G. P., Gavidia D., Gamage, C. and van Steen, M. (2006) A Secure Peer Sampling Service. UBLCS-2006-17, University of Bologna, Dept. of Computer Science.

[10] Locher, T., Moor, P., Schmid, S. and Wattenhofer, R. (2006) Free Riding in BitTorrent is Cheap. 5th Workshop on Hot Topics in Networks (HotNets), Irvine, California, USA, November 2006. http://dcg.ethz.ch/projects/bitthief/

[11] Picconi, F. et al (forthcoming) Multi-Swarm Dynamics for Fairness in BitTorrent. Technical Report, University of Bologna, Dept. of Computer Science.

[12] Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., Venkataramani, A. (2007) Do incentives build robustness in BitTorrent? 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007). http://bittyrant.cs.washington.edu/

[13] Voulgaris, S., Gavidia, D., and van Steen, M. (2005) CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. J. Network Syst. Manage., 13(2).

[14] Peersim Peer-to-Peer Simulator, http://peersim.sf.net