A Framework for a Comprehensive Evaluation of Ant-Inspired Peer-to-Peer Protocols

Amos Brocco and Ingmar Baumgart
Institute of Telematics,
Karlsruhe Institute of Technology, Germany
Email: amos.brocco@gmail.com, baumgart@kit.edu

***Abstract*—Following a constant rise in the complexity and scale of peer-to-peer networks, researchers have looked at biological phenomena in order to develop self-organized, adaptive, and robust management systems. Our focus is on distributed swarm intelligence mechanisms that mimic the behavior of social insects to solve problems such as overlay management, routing, task allocation, and resource discovery. A central problem in the validation of novel networking solutions is their empirical evaluation under different conditions. Whereas existing network simulation platforms lack specific support for ant-inspired protocols (like transparent agent migration), dedicated frameworks for bio-inspired systems fail to implement accurate network models. To bridge this gap, we introduce a framework with support for bio-inspired techniques and realistic network underlay simulation based on OverSim. To validate our work, we describe the implementation of several swarm-based protocols and we provide some measurements of the simulation performance.**

I. Introduction

Nowadays distributed systems are characterized by their large scale, complexity, heterogeneity, and dynamic nature. In this context, conventional paradigms yield several issues for efficient deployment and management, and self-organized solutions have been investigated. Among these approaches, bio-inspired ones have found their way in an growing set of network related problems [**?**]. For example, the observation of the collective behaviors of insect colonies has led to the development of fully distributed solutions that rely on interactions between simple individual agents to achieve routing or resource discovery in large scale networks. Biological systems are typically promoted as simple, intrinsically robust, resilient, adaptive, and self-organized. Although these claims might be valid for real biological systems and processes, coming up with valid bio-inspired networking solutions is not an easy task, as the latter operate in different conditions and are subject to different constraints (such as communication efficiency or computational complexity) than the former. Accordingly, in this paper we focus on the problem of validating peer-to-peer protocols based on the bio-inspired paradigm of social insects, namely ant colonies.

Among the existing swarm protocols, notable examples are: AntNet [**?**], which solves the routing problem by replicating the ant foraging process, Self-Chord [**?**] and Self-CAN [**?**] which reorganize resources to improve existing structured overlays, Messor [**?**], which implements a fully distributed load-balancing mechanism, SemAnt [**?**], a resource discovery protocols based on pheromone trails, and BlåtAnt [**?**], a self-structured overlay maintained through the collaborative behavior of ant-like mobile agents. When some new network protocol is proposed, researchers are expected to come up with experimental results that confirm their findings and show how these new methodologies hold up against existing approaches. In this regard, protocol analysis is of paramount importance to ensure a quick and efficient deployment of novel paradigms. Whereas for simple algorithms an analytical proof might be enough for demonstrating their qualities, for large and complex systems like peer-to-peer overlays, only real-world setups provide a comprehensive evaluation testbed. However, performing real experiments with large-scale networks is often impractical (at least at the early stage of development) due to time constraints or excessive deployment costs, and cannot ensure reproducible results [**?**].

To overcome these issues, simulators are employed to perform tests in a variety of scenarios that replicate typical network conditions. Concerning *traditional* peer-to-peer protocols, several simulators are available [**?**]. Most of these tools already implement a number of existing protocols and provide a clean API that eases protocol development and understanding. To assess the situation concerning the simulation of bio-inspired protocols, we reviewed several scientific papers proposing distributed swarm-intelligence algorithms. Out of the 36 considered publications, the majority (19) presents results obtained by means either unspecified or custom simulators. Several approaches to routing in ad-hoc networks (5) employ the QualNet [**?**] simulator. The ns-2 [**?**] simulator is also employed in 4 of the considered projects, whereas AntHill [**?**] and PeerSim [**?**] account for 3 and 2 projects respectively. Finally OMNET++ [**?**], GloMoSim [**?**], and Overlay Weaver [**?**] appear to have been employed in only one of the considered projects each. A common issue found in custom simulators used to evaluate bio-inspired protocols is the lack of accurate network simulation; moreover, several research projects fail to provide details about the estimated network overhead of the proposed solution. We argue that this heterogeneity prevents consistent validation and replication of published results, and the lack of a comprehensive support for bio-inspired features in existing simulation platforms hinders the convergence toward a common solution.

In this respect, this paper presents a novel simulation framework based on OverSim [**?**] which aims at supporting the implementation of bio-inspired protocols based on the

ant colony paradigm, and enables consistent evaluation and comparison. The remaining of this paper is organized as follows: Section II discusses related work concerning simulation platforms for bio-inspired networking protocols. Section III introduces our framework, while Section IV discusses some examples of implemented protocols. Finally, Section V reports the results of a performance evaluation, Section VI summarizes our conclusions on this work and provides some insights on future work.

II. Related Work

As highlighted by our brief survey presented in Section I, there exist several platforms for the simulation of peer-to-peer protocols. As a complete review of all existing tools is out of the scope of this paper, in this section we focus on simulation platforms that specifically address multi-agent and biologically inspired systems. For a detailed survey of traditional simulation frameworks, the interested reader can refer to [**?**], [**?**] or [**?**].

A first example, named AntHill [**?**], has been developed in the framework of the Bison project to support the development, evaluation, and deployment of bio-inspired peer-to-peer applications. Anthill supports both a cycle-based simulation of a protocol as well as distributed deployment using JXTA [**?**]. Whereas the former enables large scale simulations with thousands of nodes, the latter is suitable for evaluation in real network testbeds such as PlanetLab [**?**]. The development of the AntHill project halted in 2002, with the development focusing on the more generic PeerSim [**?**] simulator. A well known load-balancing algorithm, named Messor [**?**], has been developed using AntHill. Although AntHill does not simulate the underlay network, protocol validation under real world conditions can be achieved using the distributed testbed.

Similar to Anthill, Solenopsis [**?**] is a distributed middleware that focuses on swarm-intelligence based protocols. The platform provides a domain specific language to describe ant-agent's behavior and to control the execution on each node. The framework can operate both in simulation and in deployment mode, with the only difference being the number of virtual nodes managed by each host. Solenopsis employs an event-based simulator, and can reproduce simple communication latency by means of delayed message delivery; however, accurate underlay simulation is not supported. In contrast to the aforementioned platforms, the distributed middleware presented in [**?**] provides a general-purpose execution environment for self-organized agent systems that targets real distributed systems rather than simulations. This framework also has a narrower focus on resource provisioning: each agent can specify its requirements in terms of computational resources, and the systems provides him with a list of suitable nodes where the agents can migrate to.

Other simulation platforms concentrate on the global dynamics of multi agent systems, and are meant as a tools for studying general complex systems. For example, The Swarm Simulation System [**?**] focuses on the evaluation of coordination mechanisms within swarms: each swarm represents a collection of agents executing some scheduled actions. The platform supports hierarchical and nested models with agents composed of swarms of other types of agents. Similarly, the Multi-Agent Simulation Suite [**?**] provides a user-friendly environment for the development, simulation and analysis of agent-based systems. MASS includes graphical tools to assist users with limited programming skills in the creation of complex multi-agent systems. Finally, NetLogo [**?**] provides a generic programmable visual environment to experiment with complex systems. An extensive library of examples ranging from biology models to social networks is available. Due to its simplicity, NetLogo has been widely used to model and understand the dynamics of complex systems. Unfortunately, as with the two previous examples, no network-oriented features are supported.

To foster the use of bio-inspired techniques in distributed systems, and support the claims of increased adaptiveness and robustness, we argue that an accurate validation in real-world network conditions is essential. Comprehensive evaluation can be achieved through packet-level simulation, which involves reproducing transmission and delivery delays of each packet, queuing effects, jitter (i.e. variations of the latency), and channel bandwidth. Furthermore realistic usage conditions need to be considered, by means of simulated churn and traffic patterns: in this regard, even accurate network simulators sometimes fail to provide implementations of common churn and failure models. Concerning the simulation of bio-inspired systems, we argue that the major problem is the lack of explicit support for bio-inspired features (such as mobile agents) which complicates the implementation and analysis of complex multi-agent systems. On the other side, our review of the existing simulation platforms has highlighted the fact that current bio-inspired frameworks do not implement realistic network underlays. Our research is thus concerned with the development of a novel simulation platform to bridge the gap between accurate network simulation and bio-inspired system evaluation. In particular, we focus on bio-inspired peer-to-peer protocols based on the ant colony paradigm, as it has proven to be one of the most influential ones. The proposed solution is based on an existing simulation platform, called OverSim. OverSim already provides a rich set of features to facilitate accurate evaluation of peer-to-peer protocols, such as churn models, realistic underlays and packet-level simulation. In contrast to other simulators, such as ns-2, OverSim has the advantage of being strictly focused on P2P protocols, and thus provides a more comprehensive support for common features found in these types of distributed systems. Furthermore it comes with a graphical interface that shows the flow of message and eases protocol debugging. Finally, OverSim already implements a number of well-known P2P protocols, and thanks to its modular architecture it can be easily extended to support bio-inspired ones.

III. Framework Overview

The development of our framework has been driven by several requirements that have been deemed essential for a

comprehensive validation of networking protocols. We argue that the cornerstone of simulation is the ability to accurately reproduce the environment being simulated. In this respect, the simulation platform must support event based packet level simulation. Moreover, to ease the comparison with existing solutions, we deemed important to have many common protocols already implemented. The final requirement is flexibility and modularity, in order to support a broad range of bio-inspired applications and ease the integration of new features. Instead of developing a new platform from scratch, we opted to extend an existing and popular one, namely OverSim [**?**], as it fulfills many of our requirements and provides a modular architecture that facilitates both the implementation as well as the evaluation of peer-to-peer overlay protocols.

A. OMNET++ and OverSim

The core of the framework is based on OMNET++ [**?**], a discrete time event based simulator. OMNET++ allows for the simulation of message exchange between user defined modules. Its flexible architecture is suitable for simulating network communication, and using of a graphical interface the user can follow the flow of messages and display the topology of the system. Basic modules, which are implemented using C++, can define their response behavior that is executed upon reception of a message. Compound modules, which may group several basic modules can be easily created using a simple definition language called NED and a graphical interface. OMNET++ is a very popular tools in the field of communication networks, and several extensions have been created to support simulation of diverse types of networks, such as mobile and ad-hoc ones. On top of OMNET++, the OverSim framework introduces higher-level abstraction to facilitate the development of peer-to-peer protocols, such as RPC support, key-based routing, and churn models. OverSim is build upon a three layers architecture, comprising an underlay (with packet-level simulation), an overlay, and an application level. OverSim has been employed within several research projects and includes implementations of widely known P2P protocols.

B. Framework for bio-inspired algorithms

By reviewing the most representative publications in the field of bio-inspired network protocols, we identified the main requirements to conveniently support the development and evaluation of such algorithms, namely transparent migration and API for pheromone management. Accordingly, our framework implements such specific features to ease the development of ant-inspired protocols, while retaining the aforementioned benefits of OMNET++ and OverSim platform.

a) Strong, transparent migration: An important aspect of algorithms and protocols built upon the social insect paradigm is agent mobility. Whereas traditional protocols are defined by the logic that resides on each node and by the information exchanged between peers, swarm intelligence solutions are defined by the interaction between mobile agents and nodes mainly act as datawarehouses. Accordingly, a mobile agent abstraction has been deemed essential to facilitate the implementation of such bio-inspired systems. To support this feature, the proposed framework provides a domain specific language (DSL) to describe the behavior of each agent and enable strong transparent migration, namely transferring the agent runtime state to another node and resume its execution. The behavior of ant-inspired agents can be easily described, compiled to C++, and subsequently integrated within an OverSim module (as depicted Figure 1).

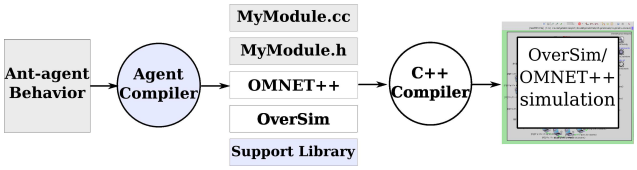

Fig. 1. Toolchain overview

The compiler generates three header files which provide methods executing the agent's behavior and the necessary prototypes that need to be included in the C++ module class definition. Because agents' behavior is compiled into a *native* C++ OverSim/OMNET++ module, the complete toolkit is only required if the behavior of the agents is modified. Figure 1 depicts the toolchain and the steps required for compiling the agents' behavior into an executable simulation. The generated C++ is not meant to be human-readable, as it contains instructions for a stack machine: a stack object (which represents the runtime state during execution of an agent) can be serialized, transmitted, and deserialized during migration. The data serialization format is platform independent and inspired by Bencode, the encoding used by BitTorrent [**?**]: this enables the implementation of runtime libraries outside OverSim and in a programming language different from C++, in order to simplify deployment of applications using bio-inspired protocols from a common DSL. A library provided with the framework includes several functions for the manipulation of the stack as well as of the available value types. In this regard, the supported types are: numbers (integers, single precision floating point, long integers, and double precision floating point), strings, lists (implemented as dynamic arrays), maps (hash-tables with string keys), and lambdas (to implement closures). To support automatic memory management each type is wrapped by a shared pointer (`shared_ptr`, as defined by C++ 0xx TR1): wrapping and unwrapping methods are provided to simplify the conversion between OverSim and agent's types. OverSim modules can define methods that can be invoked by agents, conversely C++ code can start the execution of agents.

Using natural language, the behavior of an agent is usually very concise and simple to describe, for example: *The agent migrates across the overlay at random for 5 hops. Each visited node is stored in a vector carried by the agent: after 5 hops the agent migrates back to each of the previously visited nodes*. In a traditional network simulator, like OverSim, the resulting implementation (Figure 2) could be however difficult to understand, because both the information carried by the

ant as well as its execution state needs to be encapsulated into a network packet. Furthermore, due to the lack of strong migration, we need to differentiate between 2 execution states, namely a *forward* state (where the ant collects the identifiers of the visited nodes) and a *backward* one (where the nodes are revisited in reverse order). It is clear that, if the agent's behavior becomes more complicated, the complexity of parsing an incoming message would rapidly grow and further hinder comprehension of the algorithm.

```
void Protocol::handleProtocolMessage
                (ProtocolMessage* msg)
{
    switch(msg->getState()) {
    case FORWARD:
        if (msg->getHops() == 0) {
            msg->setState(1);
        } else {
            msg->setVisitedArraySize(
                msg->getVisitedArraySize()+1);
            NodeHandle nh = getThisNode();
            msg->setVisited(
                msg->getVisitedArraySize()-1, nh);
            msg->setHops(msg->getHops() - 1);
            sendMessageToUDP(*getRandomNeighbor(), msg);
            break;
        }
    case BACKWARD:
        if (msg->getVisitedArraySize() > 0) {
            TransportAddress target =
            msg->getVisited(
                msg->getVisitedArraySize()-1);
            msg->setVisitedArraySize(
                msg->getVisitedArraySize()-1);
            sendMessageToUDP(target, msg);
        } else {
            delete msg;
        }
        break;
    default:
        delete msg;
        break;
    }
}
```

Fig. 2. Example protocol (OverSim implementation)

On the contrary, as shown in Figure 3, the proposed domain-specific language enables a high-level seamless implementation of the agent. The framework takes care of transferring the whole execution state between nodes when the `migrate` function is invoked. Support for strong transparent migration is convenient for designers during the prototyping stage, as the agent protocol (namely the information carried by the ant) may be subject to frequent changes. An additional benefit of using a custom language is a better separation of concerns between peers and agents: whereas ant-like agents define the logic of the distributed algorithm, peers can solely act as storage containers.

```
(var visitedNodes [])
(var remainingHops 5)
(while (> remainingHops 0) (begin
    (push visitedNodes (getThisNode))
    (set! remainingHops (- remainingHops 1))
    (migrate (getRandomNeighbor))))

(while (not (empty? visitedNodes)) (begin
    (migrate (pop visitedNodes))))
```

Fig. 3. Example protocol (DSL implementation)

b) Pheromone: Protocols based on the ant colony optimization (ACO) paradigm usually depend on indirect communication between individual agents, namely stigmergy. This communication pattern is usually achieved by means of pheromone trails that mimic chemical trails left in the environment by real ants. Ants may deposit pheromone on a path (node connection), to mark the direction toward a food source (resource provider). The concentration of each path can be sensed by other agents, and indicates its desirability. Chemical pheromone evaporates, but trails can be maintained by continuously *reinforcement*, which involves depositing new pheromone. To simulate artificial pheromone trails, our framework library includes a pheromone class that implements various reinforcement and evaporation models (linear, exponential, temporal); custom models can also be used. Peers are responsible for managing pheromone objects: each object can keep track of multiple trails associated with the address of nodes in the overlay. Figure 4 shows the creation of a new pheromone object using an exponential semantic, with decay factor equal to 0.1 and reinforcement factor equal to 1.5. More specifically, for a concentration τ, its new value τ' will be $\tau' \leftarrow \tau \times 0.1$, and $\tau' \leftarrow \tau \times 1.5$, after decay and reinforcement respectively.

```
PheromoneSemantic *exp =
    new ExponentialPheromoneSemantic(0.1,1.5);
PheromoneTrail *t = new PheromoneTrail(exp)
```

Fig. 4. Pheromone instance

As shown in Figure 5, an agent can sense the actual concentration of the trail associated with the path toward *someNeighbor* by invoking the `get` function of a pheromone object, and deposit new pheromone according to the reinforcement model using the `reinforce` function.

```
(var concentration (get someNeighbor))
(reinforce someNeighbor)
```

Fig. 5. Pheromone sensing and reinforcement

Peers are responsible for simulating the evaporation of pheromone trails by periodically calling the `decay` method. The semantic of pheronome evaporation depends on the selected decay model.

c) Measurements: OverSim is mainly concerned with structured overlays; as such it lacks proper support for unstructured topologies. To overcome this issue and facilitate evaluation of bio-inspired overlay management protocols that deal with unstructured networks, methods to measure topological properties such as diameter, graph cycles and girth, average path length, and clustering coefficient have been included in the framework.

IV. Case studies

Our framework inherits the modular design of OverSim, and enables the development of different types of peer-to-peer protocols. As shown in Figure 6, three protocol architectures are currently considered: stand-alone overlay, hybrid overlay,

and application. In this section we provide examples for each protocol architecture to illustrate the benefits of our framework.

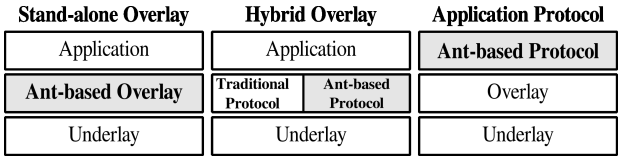

Fig. 6. Overview of supported protocol architectures

Due to space constraints we only provide a detailed discussion of the hybrid protocol, namely Self-Chord [**?**], as it represents an example that clearly leverages the library of protocols implemented by OverSim.

A. Stand-alone Overlay Protocols

Concerning the development of bio-inspired overlay protocols we implemented the BlåtAnt [**?**] algorithm. Beside maintaining logical links to connect peers together, this protocols continuously optimizes the topology in order to bound the diameter of the overlay and to reduce redundant paths. Pheromone trails, as supported by our framework, are employed to guide agents across the overlay and to detect failing peers. The operation of this algorithm can be easily evaluated in dynamic conditions thanks to the different churn generators available in OverSim, and its robustness can be compared to other overlay protocols. By means of the measurement classes the properties of the topology, such as its diameter and girth, can be easily determined.

B. Application Protocols

The last scope concerns applications running on top of an overlay. In this regard, we developed a novel task load balancing algorithm that employs ant-like mobile agents moving across a Chord [**?**] overlay. In contrast to previous examples, agents are managed independently from the overlay protocol, but can exploit overlay links to discover neighboring nodes.

C. Hybrid Overlay Protocols

A major benefit of our simulation platform is its tight integration with OverSim which facilitates the extension of existing *traditional* protocols by means of ant-based swarm intelligence. It is our opinion that novel approaches need not to replace existing solutions, but can be used to improve some aspects. Self-Chord [**?**] represents a self-organized version of the popular Chord protocol. Chord implements a distributed hash-table where information can be stored on a peer-to-peer overlay structured as a ring. Nodes and resources are assigned with unique identifiers, and each node is responsible for storing an interval of the resources key space. There is a strong link between nodes identifiers and resource identifiers, hence a key look-up operation resolves to a routing of the query toward the corresponding node. With Self-Chord this tight link is not present: resources are arranged in a self-organized way on the ring by means of ant-like agents. Each agent moves resource identifiers as to maximize the similarity among identifiers stored on the same node and achieve a global ordering on the ring. Self-Chord thus implements a different storage and lookup mechanism than Chord, and strives to improve balancing over the overlay and the overall robustness of the system.

1) Overlay Management: Overlay management is achieved using the Chord protocol. Chord employs a ring structured topology; each node is assigned a unique identifier randomly chosen from a 160 bit key space. The protocol maintains a global ordering of the nodes on the ring following their identifier. Each node has a successor and predecessor on the ring, and maintains a finger table with the addresses of additional nodes to quickly forward messages across many hops.

2) Resource Management: Whereas the traditional Chord protocol maps resources to unique identifiers in the same key space as nodes, with Self-Chord such link is not required. Each node computes an *average value* of its resource identifiers and that of neighboring peers. This average, called centroid is used by agents as a reference for clustering resource identifiers in a self-organized way: in a stable overlay, centroids are ordered along the ring.

3) Bio-inspired agents: Bio inspired agents are periodically started on each node and wander across the ring. On each visited peer, the agent looks for the resource identifier which has the least similarity with the current centroid. The agents picks such identifiers with a probability proportional to the distance between it and the centroid. Subsequently, the direction of the agent on the ring is determined: if the identifier is greater than the centroid, the agent continues by migrating toward the successor on the ring, otherwise the agent goes to the predecessor. This behavior results in an ordering of the centroids on the ring, which is essential for the look-up process to work.

Figure 7 lists an excerpt from the actual implementation using the agent programming language. The body of the agent is a `while` loop that is repeated as long as the maximum number of steps (hops in the overlay) is not reached and the agent is not carrying any resource. The `doPick` and `doDrop` functions are used to implement the aforementioned behavior. The `shouldDrop` function (not detailed in the example code) is used to determine if the carried object is to be dropped on the current node, according to a probability proportional to the similarity between the resource and the local centroid. Conversely, the `shouldPick` function determines if a resource is to be picked up by the agent. Agents can migrate in two ways: linearly and logarithmically. In linear mode agents follow the predecessor and successor links on the ring in order to reach a suitable node for dropping. Conversely, in logarithmic mode, the addresses stored in the finger table are employed to quickly reach distant peers. The logarithmic mode helps speeding up the convergence of the system toward a stable ordering of the resource keys on the ring; on the other hand, linear mode is less susceptible to instabilities and is more suitable for systems that are almost ordered. Nodes start by creating agents in

```
(define (doPick) (synchronized
    (var c (getCentroid))
    (foreach r in (getResources) (begin
        (if (and
                (shouldPickA c r direction)
                (shouldPickB c r)) (begin
            (set! resource (pick (key r)))
            r
            (break)))))))

(define (doDrop) (synchronized
    (if (shouldDrop (getCentroid) resource) (begin
        (drop resource)
        (set! resource nil)
        (end)))))

(while 1 (begin
    (if resource (doDrop) else (doPick))
    (if (= step 0)
        (if (not resource)(end))
    else
        (set! step (- step 1)))
    (if (and logarithmicHopping resource) (begin
        (migrate (getNextHop (key resource))))
    else (begin
        (if (= direction LEFT)
            (migrate (getPredecessor))
        else
            (migrate (getSuccessor)))))))
```

Fig. 7. Excerpt from the Self-Chord agent

logarithmic mode; when a node detects that the network is stable enough, the following agents are instanced with linear mode.

4) Look-up process: The look-up process of Self-Chord is very different from the original Chord protocol. Whereas in the latter a routing toward the peer whose identifier matches that of the queried resource is enough to determine a path in the overlay, the self-organized nature of Self-Chord requires a different approach. First, the direction of the query (either forward or backward in the ring) is determined by comparing the query with the centroid on the current node. Subsequently, by determining the difference between consecutive centroids in the neighborhood of the peer, the node computes the number of steps that are necessary to reach the target peer and the node either uses the successor and predecessor links or the finger table.

V. Benchmarking

In the previous section we reported on the different types of protocols supported by our framework. In order to evaluate the simulation performance and the scalability of the framework we consider here the basic multi-agent protocol presented in Figure 3 of Section III. We execute the protocol on simple overlay network, and compare the total simulation time and memory requirements with that of OverSim and of PeerSim [**?**] (a Java based peer-to-peer simulator). Each node periodically (every 2 seconds) deploys an agent and sends it to a random node with a probability of 10%. Each simulation run represents 6 hours of network activity, and for simplicity we considered a random overlay where each node knows some other nodes present in the network. Experiments are performed on a machine running Ubuntu Linux 8.04 (64 bit) equipped with an Intel Core 2 Duo L7500 processor at 1.6 GHz and 2 GB of memory.

In OverSim we employ an underlay model based on Internet latency measurements, with an average latency between peers of about 100ms. Conversely, with PeerSim a simplistic underlay model with random latency between 50ms and 150ms is considered. In contrast to OverSim, PeerSim neither models queueing nor bandwidth effects. Figure 8 illustrates the total simulation time for simulating a peer-to-peer network with different sizes (100, 1000, and 10000 nodes; *bio-inspired framework* referring to the platform presented in this paper). PeerSim benefits from its simple underlay model, and is the fastest among the three simulators, running 7 to 12 faster than OverSim. Compared to the latter, the additional abstraction layer introduced by our bio-inspired framework to achieve transparent migration further reduces the performance by approximately a factor of 2.8 in the test setup; still our framework enables faster than realtime simulations, with a speed-up factor of 4 in the 10000 node experiment. The slowdown in both OverSim and our framework can be attributed to the additional time spent for memory allocations as the network size is increased.

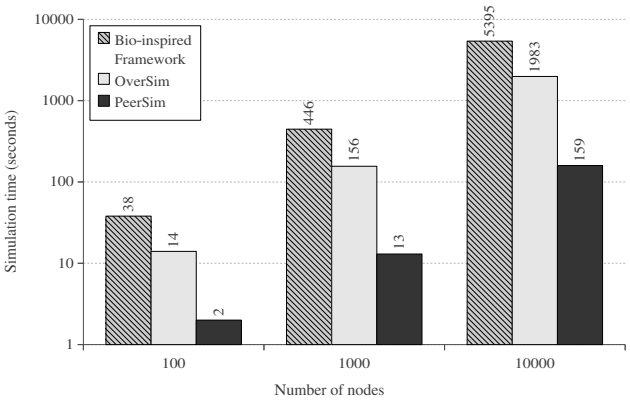

Fig. 8. Example Protocol - Simulation Time

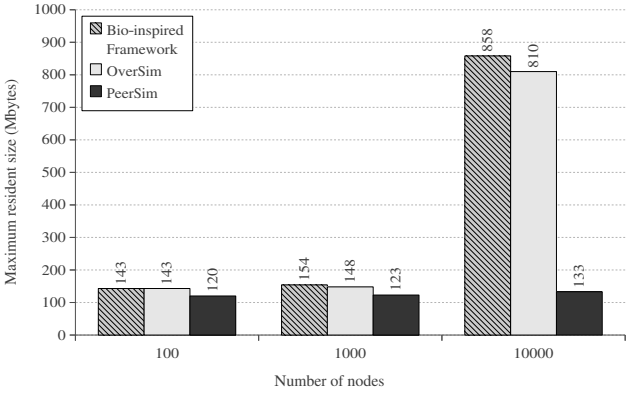

Fig. 9. Example Protocol - Memory Usage

Concerning memory (Figure 9), PeerSim shows the lowest consumption because of its simplified simulation model.

Whereas PeerSim message *transmission* only involves passing a reference to an object between nodes, in OverSim full message serialization and packet encapsulation (and hence data duplication) is required to achieve accurate simulation of the underlying network. Our bio-inspired framework performs on par with OverSim. The huge difference between simulations with 100 and 1000 nodes, and 10000 nodes can be explained by the fact that at small scales the memory occupied by nodes is relatively small compared to the rest of simulation environment, and thus fits within the default allocation of the framework (about 150 MBytes). These benchmarks shows that both the bio-inspired features and the accurate underlay model of our framework introduce a sensible overhead, nonetheless large simulations in realtime are still feasible. We should stress that this overhead is outweighed by the benefits introduced by strong transparent migration support and a high-level agent description language which simplify the prototyping and readability of ant-inspired protocols. In this regard, the additional features provided by our framework alongside with OverSim (like a graphical user interface, detailed statistics, and modeling of queueing effects) greatly facilitate rapid development and evaluation.

VI. Conclusion

In this paper we presented a novel framework for the development and evaluation of swarm-based peer-to-peer protocols. In contrast to existing solutions our approach allows for accurate simulation of different network aspects, such as transmission delays and churn, and integrates with the OverSim/OMNET++ platform. Because of this highly modular underlying platform that already implements several protocols, researchers can easily compare novel approaches with existing ones. Amongst the benefits of our platform over existing solutions is the availability of a high-level programming language which enables seamless implementation of agent-based protocols, as well an API for pheromone management and topology measurements. Our current research focuses on the evaluation of several overlay and application-level protocols and their comparison with traditional approaches that are already implemented by OverSim. Future work includes optimization of the runtime performance, validation of swarm based underlay routing and experiments with mobile and ad-hoc networks. Furthermore, other bio-inspired techniques such as epidemic protocols and genetic algorithms will be investigated. More generally, our goal is to investigate the implementation of different bio-inspired patterns, such as the ones presented in [**?**]. Our framework is actively developed and released under an open source license; the source code is available at http://www.syscall.org/doku.php/overswarm.

VII. Acknowledgments

This research has been carried out thanks to the financial support of the Swiss National Science Foundation, fellowship Nr. 134285.