Karlsruhe Institute of Technology (KIT)
Institute of Telematics

TELEMATICS TECHNICAL REPORTS

OverSwarm: a simulation tool for biologically inspired peer-to-peer networks

Amos Brocco
Institute of Telematics, Karlsruhe Institute of Technology (KIT), Germany
brocco@kit.edu

August, 3rd 2011

TM-2011-4

ISSN 1613-849X

http://doc.tm.uka.de/tr/

Institute of Telematics
Karlsruhe Institute of Technology (KIT)
Zirkel 2, 76131 Karlsruhe, Germany

OverSwarm: a simulation tool for biologically inspired peer-to-peer networks

Amos Brocco
Telematics Institute
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
Email: brocco@tm.uka.de

***Abstract*—The complexity of today's peer-to-peer networks has led researches toward novel self-organized approaches in order to provide robust and scalable management solutions. Among them, a promising direction is represented by biologically inspired systems, which mimic the behavior of natural systems such as insect colonies or swarms of animals (birds, bees, etc.). In this paper we focus on the evaluation of bio-inspired peer-to-peer systems built upon the ant colony paradigm, by presenting a novel framework named OverSwarm. OverSwarm delivers dedicated support for bio-inspired features such as transparent agent migration (to mimic the behavior of social insects) and pheromone trails (to support indirect communication between agents). Our framework is based on the popular OverSim simulator, which provides several implementations of the most current peer-to-peer protocols. To validate our solution, different case studies are discussed, and a benchmarking is performed to evaluate the performance of our platform.**

I. Introduction

The increasing scale, dynamicity, and heterogeneity of today's distributed systems has forced researchers to reconsider conventional protocols and architectures and investigate novel self-organizing solutions to reduce management complexity and costs. Self-organization refers to the ability of a system to autonomously organize its structure in order to efficiently fulfill some function [1]. Organization should emerge spontaneously from interactions between distributed entities, without central supervision or control [2], [3]. Nature provides a wide range of examples of complex systems where simple interactions between individuals are enough to provide a global coherent behavior. Computer scientist have been fascinated by such simple designs that promise scalable and robust operation, and have tried to replicate the observed behaviors in order to bring these desirable features in the realm of computer networks. Algorithms and techniques that replicate some natural phenomena are typically referred to as bio-inspired, and several examples such as genetic algorithms or swarm-intelligence [4] have successfully been employed to solve complex optimization problems. Concerning computer networks, bio-inspired solutions have been devised for routing problems, resource discovery, information clustering and overlay management. Due to the inherent distributed nature of these solutions, peer-to-peer communication is well suited for their implementation. A widely employed bio-inspired paradigm that has been applied to computer networks is the one mimicking social insects such as ants and bees [5]. In this regard, the network represents the environment where insect-like software agents live, each node represents a nest that accepts incoming agents and might generate new agents, and agents themselves are represented by messages exchanged between nodes. As with other network applications, the behavior of nodes and the exchanged information is defined by means of a communication protocol. However, in contrast to more traditional protocols [1], insect-based bio-inspired protocols are tightly related to the concept of mobile agent [6]: whereas in the former the whole logic is implemented inside each node, and the communication protocol defines what type of information is exchanged, in the latter the messages themselves represent functional entities that are able to actively operate and move on the network. This paradigm better mimics real-life insects, which have their own intelligence and can perform more complex operation than just carry information or resources around, but is quite different from traditional networking approaches. In the nature, years of evolution have ensured that biological processes that have survived up to our days are robust and reliable enough to sustain life (unsuitable processes having disappeared). For example, the seemingly chaotic behavior of an ant nest is in fact self-organized and ordered and ensures the survival of the colony by taking care of the foraging process, the reproduction, and the defense of the territory. On the contrary, bio-inspired systems need to be verified and validated to ensure that the desired goals are achieved: whereas a mechanism might work for real insects, the technical limitations and environmental differences of a computer system might reveal unexpected deficiencies and hinder its robustness, reliability or efficiency. An essential aspect in the development and validation of novel protocols is thus a detailed analysis of their performance and behavior under real-world conditions. For example, a routing protocol must be able to compute and maintain optimal paths across the network, overcome node or link failures, and ensure an acceptable delivery rate. Unfortunately, due to technical or time limitations, it is not always feasible to carry out exhaustive experimentation on large scale networks composed of thousand of nodes or to recreate particular failure situations,

[1] In this paper we use the term *traditional* to refer to protocols and algorithms that are not explicitly based on bio-inspired principles, such as Chord or Kademlia.

such as the ones involving high churn rates, simultaneous disconnection of a large part of the network, or denial of service attacks. A rigorous proof of the robustness and performance of a network, namely by means of analytical models, involves the creation of a mathematical representation of the system. This methodology is often impractical due to the complexity of the problem and the difficulty of including all the details of real-world systems. In this regard, network simulators represent a valid alternative to verify the correctness and robustness of a network algorithm under controlled conditions. It is important to note that different validation methodologies do not rule out each other. An ideal situation involves the creation of a mathematical model, its verification through simulations, and a final validation using a real-world testbed. The use of simulators is nonetheless an important step in the evaluation of a system. In contrast to a real-world testbed, simulation tools offer benefits such as reproducibility of results and the possibility of performing automated tests. However, as pointed out in [7], not all existing simulators are able to accurately and realistically reproduce the characteristics of a computer network, and differences in the outcome of the experiments can be observed. Furthermore, existing solutions lack support for bio-inspired approaches, such as the ones replicating the behavior of ant colonies. To shed some light on how evaluation of bio-inspired protocols is currently performed, we conducted a brief survey of several research papers. We focused on 36 scientific publications that present biologically inspired approaches to networking problems ranging from routing, to resource discovery and information clustering. In 19 of the papers the simulation tool is not specified, which often means that a custom simulator was employed; details of the accuracy of the simulation (such as, for example, whether a packet level simulator was used) were also omitted. For wireless ad-hoc networks 5 papers employed the QualNet [8] simulator. The use of the widely known ns-2 simulator was reported in 4 papers, whereas AntHill [9] and PeerSim [10] were cited in 3 and 2 papers respectively. Finally, OMNeT++ [11], GloMoSim [12], and Overlay Weaver [13], have been referred by just one paper each. From these results it is clear that the effort required to conduct a comprehensive comparison of different protocols is considerable, as this heterogeneity hinders consistent evaluation of the proposed solutions and prevents results published by different authors from being compared. In this regard, we identified two main issues with current evaluation methods: first is the lack of a widely used platform that provides reference implementations for common bio-inspired and traditional protocols, second is the limited accuracy exhibited by many available simulators when it comes to simulate network traffic effects such as delays and jitter. The need for a common evaluation platform mostly arises from the observation that the majority of research project in the field of distributed swarm intelligence are still being evaluated using custom or non-specified simulation tools. Accordingly, our research is concerned with providing a tool to help researcher validate bio-inspired protocols that employ ant-like agents. Consequently, we present here the OverSwarm framework, which builds on the popular OverSim platform and enhances it in order to facilitate the development of mobile agent based systems. Our software takes advantage from several traditional protocols and an extensive simulation API that are already implemented in OverSim, as well as from the modular architecture and graphical tools of the underlying OMNeT++ discrete event simulator. In this paper we detail the features of OverSwarm and present different case studies that validate its functionalities and highlight its benefits.

The rest of this paper is organized as follows: Section III reviews several simulation tools and highlights their main features. Section IV presents the architecture of the OverSwarm framework and details its relation with OverSim. Sections V to VII detail the implementation of three different protocols to highlight the benefits of our framework and present the development methodology behind our work. Section VIII provides an initial evaluation of the framework, in terms of simulation performance and memory consumption. Finally, Section IX summarizes the findings of this paper and provides some hints on future improvements of OverSwarm.

II. Design Goals

The development of OverSwarm has been driven by the following objectives:

- Packet-level simulation: it enables a high-degree of accuracy, and can be achieved by means of a discrete event-based simulator that emulates packet-flows and replicates communication issues such latencies, jitter, packet lost, and queuing effects is necessary. Packet-level simulations allow for a comprehensive analysis of the network overhead generated by a protocol.
- Modular architecture: facilitates the development of novel protocols by reducing the effort required for understanding its inner-workings and to replace existing components with custom ones. The possibility of reusing, extending and modifying existing modules with minimal impact on the rest of the platform is also desired. A highly desired feature concerns interchangeable underlay models, which enables experimentation with different types of networks (wired and wireless).
- Common API: reduces code duplication, and simplifies the validation of novel protocols and their comparison with other solutions. In particular, a generic solution for evaluating key-based routing protocols is required. Furthermore, bio-inspired specific features, such as pheromone management, must be available to ease the implementation of swarm-intelligence based protocols.
- Protocol library: eases the comparison with traditional protocols, namely by providing ready to use implementation of the most known protocols such as Chord [14], Pastry [15], Kademlia [16], etc. This requirement is especially important to determine the advantages and drawbacks of bio-inspired solutions compared to traditional approaches.
- Visualization tools: to fully understand the behavior of a protocol, visualization tools should be provided to

display the information flow across the network during the simulation.
- Scalability: the simulation must support large-scale systems with thousands of nodes. In this regard, the simulation tool must limit its memory footprint and provide an efficient use of the available resources.
- Simulation control: enables the user to stop and resume the simulation in order to inspect the network and dynamically change configuration parameters.

III. Related Work

In this section we review a number of existing network simulation tools, and we highlight their features. Although our work is mainly concerned with swarm intelligence protocols, general simulation platforms will also be discussed. For this purpose, we distinguish between simulators that focus on *traditional* network protocols, and those that provide specific support for ant-inspired protocols, namely by facilitating their implementation or by providing bio-inspired oriented features. Our review is by no means exhaustive; the interested reader is suggested to refer to more detailed surveys of network simulators such as [17], [18] or [19].

A. Focus on network protocols

The first set of simulators have a strict focus on traditional network protocols.

1) OMNeT++: OMNeT++ [11] is a highly modular discrete event simulator that has been widely used to evaluate networking systems. Several simulation frameworks have been developed to support packet level simulation of wired and wireless networks, for example INET/INETMANET [20], and MIXIM [21]. Modules are written using C++, although bindings for Java [22] and C# [23] are also available. Among the benefits of this platform are the graphical user interface and a high level description language, called NED, for the definition of compound components and their communication channels. A graphical interface enables visual feedback during the simulation and observation of the message flow between each component and can be useful in the early prototyping stages; OMNeT++ also support command line execution for automated testing. OMNeT++ can be freely used for personal and academic purposes: a commercial version, named OMNEST, is also available.

2) ns-2: The ns-2 [24] network simulator is a discrete event simulator targeting network protocols with packet-level accuracy. Each protocol is implemented using both C++ (for performance) and OTcl (for flexibility), and can operate on simulated wired or wireless networks. The main focus of ns-2 are low-level protocols such as TCP, UDP or FTP, nonetheless it is very popular for simulating application level protocols.

3) OverSim: OverSim [25] builds upon OMNeT++ to provide a complete framework for the simulation of peer-to-peer overlay networks. Several structured and unstructured overlay protocols are already implemented and can be easily compared. OverSim retains the modular architecture of OMNeT++, which facilitates module composition and extension. Several lifetime based churn models (like Weibull, Pareto, etc.) are already implemented and can be used to evaluate the robustness and resilience of peer-to-peer networks under realistic conditions; furthermore, a generic look-up mechanism based on the common API [26] eases the validation of structured overlays that implement a distributed hash-table.

4) PeerSim: PeerSim [10] is a Java based simulator focusing on peer-to-peer protocols. PeerSim supports two simulation models, namely cycle based and event based. Cycle based simulations trade accuracy for performance, and can manage very large overlays consisting of hundreds of thousands of nodes. Conversely, event based simulations provide some level of realistic packed delivery, while requiring more resources. Although event based simulations can emulate communication latency, the details of the underlying network are neglected, and the resulting simulation remains very simplistic when compared with more accurate simulation tools.

5) P2PSim: P2PSim [27] is a multi-threaded discrete event simulator for structured overlays implemented in C++. The underlay model can emulate different topologies with realistic latencies and failure models. Several peer-to-peer protocols are available, for example Chord, Kademlia, Kelips [28], etc. P2PSim does not offer any kind of graphical visualization tool to display communication between nodes during simulations. The latest version of P2PSim dates of April 2005, and development seems to have stalled. Furthermore, documentation seems to be scarce, and no detailed information about the simulator API is available.

6) Overlay Weaver: Overlay Weaver [13] is a toolkit for implementing peer-to-peer systems which focuses mainly on structured overlays such as Chord, Pastry, and Kademlia. Like OverSim, Overlay Weaver implements a clean architecture and a common API that facilitate the development of new algorithms; the system is decomposed into different layers: an application layer, a service layer (currently implementing a DHT and a multicast interface), and a routing layer (which consists of routing algorithms and low-level messaging mechanisms). The current version provides 4 messaging systems, namely TCP, UDP, single-host emulator and distributed emulator (both with simulated message delivery latencies). With single-host emulation, the simulation system provides message passing between Java threads, whereas with distributed emulation messages can be delivered to different Java virtual machines. Overlay Weaver includes a graphical tool to visualize communication between nodes. A major drawback of this simulation platform is the fact that it only supports real-time execution, as such it becomes difficult to replicate results or to control parameters during experiments.

7) PlanetSim: PlanetSim [29] is a Java framework for overlay network simulation that supports the Common API. Network protocols can be implemented following two paradigms: algorithm model and behavior model. In algorithm mode, each node embeds all the actions that will be executed throughout its lifecycle, namely from the its creation and joining of the overlay until its disconnection or failure. On the contrary, in behavior model simulations are event based, and separate

classes can be implemented to specify the action to be performed upon reception of a message. Such decoupling enables a better separation of concerns, but results in a slight increase of the simulation time compared to the algorithm model. PlanetSim currently implements only simple underlay models and aims to be an efficient peer-to-peer simulator rather than an accurate packet-level simulator; however, according to its developers, more accurate underlays can be plugged in in order to account latencies and load network models.

8) GloMoSim: The Global Mobile Information System Simulator [12], or GloMoSim, is a scalable simulation environment for wired and wireless networks. GloMoSim supports heterogeneous communication networks with asymmetric links, traditional protocols such as TCP and UDP, as well as multi-hop and ad-hoc wireless communication. Its simulation core is implemented in a parallel C-like language called PARSEC, which supports discrete-event simulation with sequential and parallel execution.

9) QualNet: QualNet [8] is a commercial derivative of GloMoSim, and provides a comprehensive simulation platform mainly targeted at wireless and ad-hoc networks. QualNet ships with visual tools that facilitate the design of mobility patterns and three-dimensional worlds were signal propagation is accurately simulated. Similar to GloMoSim, the simulation engine supports parallel execution and is implemented using the PARSEC language. Tracing and analysis tools are available to gather statistical data about simulations.

10) PeerFactSim.KOM: PeerFactSim.KOM [30] is a modular simulator for large scale peer-to-peer systems such as content distribution networks, streaming applications, or distributed hashtables. The architecture of the simulator comprises four different layers: application, overlay, transport, and network. The underlying network model can reproduce different service models with varying degrees of realism. Along with the simulator, a set of graphical tools enable the analysis of the behavior of the system and the obtained results.

B. Focus on ant-inspired protocols

Apart from the aforementioned tools, there exist few simulation platforms that explicitly focus on biologically inspired network protocols. The main drawback of existing solutions is the lack of an accurate underlay simulation, which hinders a comprehensive evaluation under realistic conditions.

1) AntHill: AntHill [9] has been developed in the framework of the Bison project to support the development, evaluation, and deployment of bio-inspired peer-to-peer applications. Distributed systems built using Anthill support execution either on a cycle-based simulator or in a real-world setup using JXTA [31]. The former enables large scale simulations with many thousands of nodes, whereas the latter is suited for deployment and evaluation in real network testbeds such as PlanetLab [32]. The development of the AntHill project was stopped in 2002 to focus on the more generic PeerSim simulator. A well known load-balancing algorithm, named Messor [33], has been developed using AntHill: ant-like agents start from overloaded nodes and wander on the peer-to-peer overlay to discover least loaded peers and initiate a balancing operation. AntHill does not simulate the underlay network, lowering the accuracy of the simulation. On the other hand, such simplicity enables fast simulation and thus large scale experiments with thousand of nodes.

2) Solenopsis: Solenopsis [34] 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 does not distinguish between simulation and deployment mode. The only difference relies in the number of virtual nodes that area managed by a host: in simulation mode several nodes are typically run on a single host, whereas in deployment mode a one-to-one mapping is common. Communication latency can be introduced by means of delayed message delivery. The framework presented in this paper inherits some features introduced by Solenopsis, such as the focus on ant-based protocols and the agent description language.

3) Test-bed platform for bio-inspired distributed systems: The distributed middleware presented in [35] provides a general-purpose execution environment for self-organized agent systems. In contrast to the aforementioned platforms the focus of this test-bed is put on real distributed systems instead of simulations. Agents can migrate between nodes and several deployment policies are implemented. Each agent can specify its requirements in terms of computational resources, and the systems provides him with a list of suitable nodes.

4) Swarm Simulation System: The Swarm Simulation System [36] focuses on the evaluation of coordination mechanisms in multi-agent systems. Swarms represent complex adaptive systems; each swarm consists of 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. The system is meant as a tool for studying general complex systems, and lacks support for networking systems simulation.

5) MASS: The Multi-Agent Simulation Suite [37] 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. No network-oriented features are available.

6) NetLogo: Like the previous two examples, NetLogo [38] is not a network-oriented simulator, but 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. NetLogo can be used as a tool to understand the dynamics of complex systems.

C. Discussion

An essential feature for achieving an accurate validation of network protocols and realistic results is 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. Moreover, in order to

replicate realistic usage conditions, churn and traffic patterns should be simulated: in this respect, many of the reviewed simulators also fail to provide detailed statistics about churn, node failures, and message delivery rates. Concerning the simulation of bio-inspired systems, it is our opinion that explicit support for mobile agents by the simulation platform facilitates the implementation and analysis of complex multi-agent systems. Our review of the existing simulation platforms has highlighted the lack of realistic network conditions in swarm oriented tools, and missing support for mobile agent protocols in network oriented ones. It is clear that achieving a higher level of realism has an important impact on the performance of the simulator, nonetheless we argue that an optimal trade-off between accurate results and simulation overhead is possible and should be attained. Hence, with respect to the requirements detailed in the previous section, our research goal is the development of such kind of tool, to provide both accurate results in terms of network simulation and ease of use concerning complex system development. To achieve our goal we base our solution on an existing platform, having determined that the one that mostly fulfills the aforementioned requirements them is OverSim [25]. 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. Furthermore it's modular architecture is easily extended to support bio-inspired protocols.

IV. OverSwarm Framework

The OverSwarm platform focuses on facilitating the implementation and evaluation of peer-to-peer systems based on the ant colony paradigm. In contrast to other platforms with similar goals, OverSwarm is not a stand-alone product, but extends a popular and widely used peer-to-peer simulator called OverSim. Our choice not to develop from scratch is motivated by the fact that OverSim already includes a set of tools that eases the validation of new protocols, such as a protocol library that implements a wide range of protocols (Chord [14], Kademlia [16], etc.) and an API to simulate churn and resource discovery. In this section we provide a detailed view of the framework and highlight its design choices and benefits.

A. Integration with OverSim

OverSwarm has been designed to integrate well with the OverSim architecture while remaining a separate library. None of the existing modules require modification of the OverSim core library in order to enable the execution of swarm based protocols. Figure 1 illustrates the architecture of OverSim. The platform is divided into three functional layers, namely the application, the overlay, and the underlay levels. The application layer is further divided into three tiers that implement high-level services such as XML-RPC. Both structured and unstructured overlays are supported, although a major focus is put on the former, as underlined by the Common API interface between the top-most layers of OverSim. Several underlay models are available, namely INET and ReaSE [39] (to simulate network with realistic latencies and traffic patterns), Simple (which replicates latencies based on Internet measurements), and Single Host (used for real world deployments). OverSwarm currently targets both the overlay and the application layers.

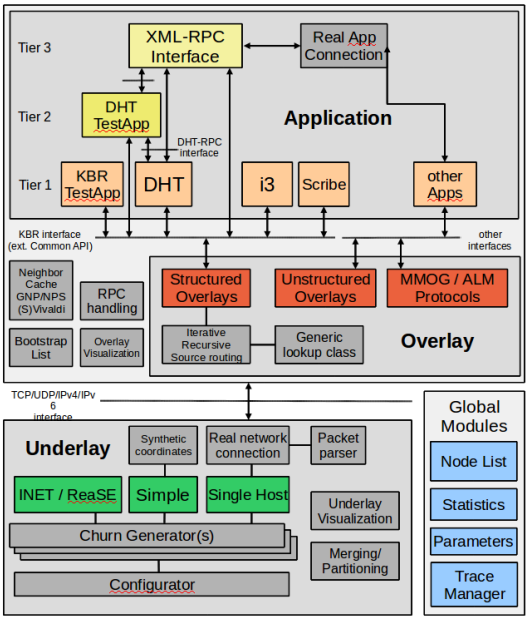

Fig. 1. OverSim architecture (reproduced under permission from the authors)

1) Stand-alone overlay protocols: OverSwarm can be used to develop stand-alone overlay protocols entirely based on the paradigm of insects colonies. To facilitate the development of such protocols a template class, which defines a skeleton module, is available. An example is presented in the following section.

2) Hybrid overlay protocols: Hybrid solutions combine traditional peer-to-peer protocols, such as Chord, and novel bio-inspired mechanisms based on ant-like agents. One such examples is Self-Chord [40], which will be discussed in detail in the following of this paper.

3) Application protocols: Application protocols execute on top of an existing overlay and can exploit the functionalities of the latter such as topology management or key-based routing.

B. Separation of concerns

Mobile, insect-like agents operate separately from nodes, and their behavior could define a completely different communication protocol. Accordingly, OverSwarm is based on a clear separation of concerns and a clear distinction between the activities of a mobile agent and those of a node. To achieve this, the implementation of agents is defined separately from C++ modules using a domain specific language similar to Lisp. Ant agents can interact with nodes, and invoke methods exported from C++ in order to access the resources made available by each peer. This decoupling simplifies the development and prototyping of protocols, because agents' behavior does not interweave with the operation of a node,

and facilitates the implementation of bio-inspired mechanisms alongside with traditional protocols. The high-level nature of the language abstracts from the low-level concerns of C++ such as memory management. Moreover, from a future development perspective, this choice enables multiple targets to be generated from a single protocol definition.

C. Toolchain

For performance reasons the agent behavior is not interpreted, but compiled to C++ instead. For this purpose, the OverSwarm's toolchain comprises a compiler that translates the description of an agent to C++ code that can be embedded into an OverSim module. In order to support strong and transparent migration the generated C++ code contains instructions that operate on a stack object (whose implementation is part of OverSwarm's library). Upon migration, the stack is serialized and transferred to the target node; if simulations are run on a single machine, OverSwarm also supports a migration procedure that does not require the serialization of the agent state, resulting in better overall performance.

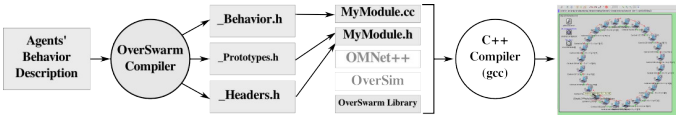

Fig. 2. OverSwarm toolchain

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 OverSwarm toolkit is only required if the behavior of the agents is modified. Figure 2 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 serialization format is platform independent, and is based on Bencode, the encoding used by BitTorrent [41]; accordingly, it is technically possible to generate code in a language different from C++, provided that a re-implementation of the core framework libraries in the target language is available. A library provided with OverSwarm 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). Automatic memory management is achieved by wrapping each type using a shared pointer (`shared_ptr`, as defined by C++ 0xx TR1): wrapping and unwrapping methods are provided to simplify the conversion between OverSim and OverSwarm types. OverSim modules can define methods that can be invoked by agents, conversely C++ code can start the execution of agents.

D. Bio-inspired related features

OverSwarm provides several features to facilitate the implementation of bio-inspired protocols.

a) Strong, transparent migration: Protocols based on the paradigm of social insects depend on the behavior of each individual (in our case a software agent) and its interaction with other individuals or the environment. In contrast, traditional network protocols are defined by the operation of each node and by the messages exchanged between peers. Existing simulation framework, such as OverSim, are based on a programming model that focus on the implementation of the latter type of protocols; as such, the implementation of agent-based biologically inspired protocols can be cumbersome. In this regard, an essential feature that enables seamless development of ant-inspired protocols is strong and transparent migration of agents. On one hand, strong migration is concerned with transferring the whole runtime state of an agent from one node to another: the system must be able to suspend the agent's execution, transfer all the necessary data to the target peer, restore the runtime state, and finally resume execution. On the other hand, transparent migration facilitates the development of an agent by removing the need for explicit serialization and deserialization of any local data maintained by the agent. OverSwarm implements these features and makes them available through primitives of the domain specific language used to program agents' behavior. Support for strong, transparent migration motivated our decision of implementing a domain specific language to describe the behavior of ant agents.

b) Pheromone management: An essential requirement concerns the availability of generic classes and an API fulfilling the basic task of managing pheromone trails. Pheromone trails replicate the indirect communication mechanism (called *stigmergy*) used by real ants which leave chemicals in the environment to signal paths toward food sources. Other individuals in the colony can sense existing paths and decide to follow them and/or reinforce their concentration. Pheromone evaporates with time, and completely disappears unless reinforcement takes place. A number of ant-inspired software systems employ artificial pheromones to achieve their aim, accordingly it makes sense to provide support within the framework. OverSwarm provides a generic and flexible `Pheromone` class that supports different reinforcement and evaporation patterns, such as linear and exponential. Additional patterns, such as the ones discussed in [42], can be easily implemented.

E. Other features

Beside bio-inspired features, OverSwarm introduces some utility functions that ease the evaluation of protocols for unstructured networks. In particular, methods to compute some topological properties such as the diameter, radius, average path length, and clustering coefficient of the overlay have been implemented.

F. Protocol definition

The behavior of the agents implements a communication protocol that is defined is an `.osw` file, as shown in the example in Figure 3. The preamble contains the protocol name, which must match the name of the class where the generated C++ will be included.

```
protocol "Foo" {
   import for "oversim" {
      "trail->reinforce" as "reinforce"
   }
   ant "Bar" {
      behavior {
         (reinforce "trail1")
      }
   }
}
```

Fig. 3. Example protocol definition

Within the `import` section the mapping between C++ methods and Lisp ones can be made. In the example, the method `reinforce` of the object referenced by the class field `trail` can be invoked by calling the `reinforce` function. Methods defined within the C++ module that can be accessed by the agent must implement a valid signature. Any number of ants can be defined by means of `ant` blocks: each ant has its own name and can define its own behavior.

V. Case Study: Self-Chord

OverSwarm benefits from its seamless integration with OverSim which simplifies the extension of existing *traditional* protocols with bio-inspired features by means of ant-inspired mobile agents. In this context, existing approaches can be complemented and improved instead of being completely reimplemented following a new paradigm. A good example of this methodology is Self-Chord [40], which provides a self-organized extension to the popular Chord protocol. Chord is a peer-to-peer protocol that implements a distributed hash-table, with information stored in a ring-structured overlay. Nodes and resources are assigned with unique identifiers (*hashes* of their address or content), and each node is responsible for managing an interval of the resources key space. Identifiers of nodes and of resources are tightly related, hence a key look-up operation resolves to a routing of the query toward the corresponding node. With Self-Chord this strong link is avoided, as resources are arranged in a self-organized way on the available peers by means of ant-like agents. Each agent moves resource identifiers as to maximize the similarity among identifiers stored on the same node. In the end, a global ordering of the resource identifiers is achieved on the ring. Self-Chord exploits a large part of the original Chord protocol (for example for overlay maintenance), but implements a different storage and look-up mechanism that aims at improving balancing over the overlay and at increasing the robustness of the system.

A. Overlay Management

The overlay management remains unchanged, and is achieved using the original Chord protocol. Chord employs a ring structured topology where 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 according to their identifier. Each node knows its successor and predecessor on the ring, and maintains a finger table with the addresses of additional nodes to quickly forward messages across many hops (at increasing distances).

B. Resource Management

In contrast to the traditional Chord protocol, Self-Chord does not map resources to unique identifiers. More specifically, resource identifiers can be defined over a different key space than nodes. To determine where a resource should be stored, each node computes an *average value* of the identifiers of the resources already present on the node and on 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 and an equal distance can be observed between centroids on consecutive peers.

C. Bio-inspired agents

Each node periodically creates new bio inspired agents that wander across the ring. On each visited peer, the agent looks for the resource identifier which has the least similarity with the current centroid. With a probability proportional to the distance between it and the centroid, the agents picks one of the resources. The agent tries to relocate the resource to a node whose centroid is as close as possible as the resource identifier. Accordingly, the direction of the agent on the ring is determined as follows: 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. Although each agent operates in an independent manner, the global behavior of all agents results in a global ordering of the centroids on the ring. This ordering is essential for the look-up process to work, as queries need to estimate the position of the resource by computing the number of hops in the overlay based on the difference between consecutive centroids.

The actual implementation using OverSwarm's agent programming language reflects the aforementioned description. Figure 4 lists an excerpt of the code. 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 pick up by the agent. In order to speed up the ordering in the overlay initialization phase, 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,

```
(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 LOGARITHMIC_HOPPING resource) (begin
        (migrate (getNextHop (key resource))))
    else (begin
        (if (= direction LEFT)
            (migrate (getPredecessor))
        else
            (migrate (getSuccessor)))))))
```

Fig. 4. Excerpt from the Self-Chord agent

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. For these reasons, nodes start by creating agents that travel in logarithmic mode; when a node detects that the network is stable enough, agents are instanced with linear mode in order to avoid instabilities.

D. Look-up process

Because resource identifiers are not tied with peers' identifier, the look-up process of Self-Chord is completely 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 the node determines the number of steps that are necessary to reach the target peer and the node, by computing the difference between consecutive centroids in the neighborhood of the peer. Finally, the target peer is chosen using either the successor and predecessor links or the finger table. If the target is not reached, further forwarding following the same principle could be performed.

VI. Case Study: BlåtAnt

BlåtAnt [43] is an overlay management protocol which aims at maintaining an optimized topology to reduce the cost of broadcasting a message to all peers. BlåtAnt employs different species of ant-like mobile agents that are assigned with different tasks such as discovering distant nodes, connecting and disconnecting peers, or signaling the liveness status of a node. The overlay maintained by this protocol can be classified as self-structured, because the topology is maintained in an adaptive way in order to fulfill certain criteria, namely an upper bounded diameter and a lower bounded girth (length of the smallest cycle). In this section we consider the -S variant of the algorithm, as detailed in [44].

A. Optimization process

The overlay optimization process is controlled by two simple rules that determine the creation of new logical links and the destruction of existing ones. The goal of the link creation phase is to limit the diameter of the network according to an optimization parameter D. More specifically, a new link is created between two peers if a distance greater than

$$2D - 1$$

is observed. Conversely, removing links aims at breaking up cycles in the topology that contribute in increasing the retransmission rate when the overlay is flooded with queries. The rule for breaking up cycles states that two neighbor peers must be disconnected if there exist an alternative path that connects them whose length is less than

$$2D - 3$$

hops. Observation of distances between peers as well as connection and disconnection are performed by means of ant-like agents. In order to detect node failures and abrupt disconnections in high-churn situations, each agent leaves pheromone trails in the network while migrating; the concentration of the pheromone determines the liveness of nodes and enables the system to react accordingly. Two kind of trails, associated with each neighbor on each node, are employed: beta and gamma. Beta trails are reinforced by incoming ants, and their purpose is to track communication originating at neighboring nodes; if a beta trail completely evaporates a peer can deduce that the corresponding neighbor failed (i.e. abruptly disconnected) or that some communication problem exists, and initiate a recovery procedure to connect with some other peer. Conversely, gamma pheromone trails are used for two purposes: on one hand to ensure that all paths in the overlay are equally explored by Discovery agents (as such ants most likely follow the lowest concentration trails); on the other hand, low gamma concentrations trigger ping traffic on the corresponding path to signal aliveness of the node.

B. Agents

BlåtAnt employs different species of ant-like agents. We report and discuss here the behavior of the most significant ones.

1) Discovery: Discovery agents are periodically instanced on each node with some probability. Each agent wanders on the overlay for a predefined maximum number of steps: on each peer the identifier of the node (typically its IP address and port) is stored into a bounded size vector. Subsequently,

```
(while 1 (begin
    (if (<= steps 0) (break))
    (if (inform vector) (clear vector))
    (push vector (getThisNode))
    (if (> (len vector) vectorlength) (erase vector 0))
    (var neighbors (getNeighbors))
    (remove neighbors previous)
    (foreach v in vector
        (remove neighbors v))
    (if (= (len neighbors) 0) (begin
            (set! neighbors (getNeighbors))
            (set! vector [])))
    (if (< (random) kappa)
        (set! nextStep (getLowestGammaTrail neighbors))
    else
        (set! nextStep (choose neighbors)))
    (if nextStep (begin
        (set! previous (getThisNode))
        (set! steps (- steps 1))
        (if (not (migration nextStep)) (end))))))
```

Fig. 5. Discovery Agent

```
(virtual $target)
(var source (getThisNode))
(if (migrate $target) (begin
    (var d (getEstimatedDistance source))
    (if (and (> d 0) (< d (- (* 2 (D)) 1 ))) (end))
    (var isAcceptedConnection (connect source))
    (if isAcceptedConnection
        (if (migrate source)
         (connect $target)))))
```

Fig. 6. Optimization Link Agent

a candidate target node is chosen among the neighbors of the current peer, such that previously visited peers are not considered. If no candidate is available, all neighbors are considered. With a given probability k, the agent migrates to the neighbor associated with the lowest concentration of gamma pheromone, otherwise it migrates to a random neighbor.

The behavior of the agent can be easily implemented with OverSwarm, as shown in Figure 5. The `steps` variable keeps track of the current number of hops traveled in the overlay. The addresses of visited nodes are stored in a dynamic list `vector`. Each time a new address is added to the list, its size bounds are check, and exceeding information is eventually removed. Selection of the neighbor with the lowest gamma pheromone concentration is implemented as a separate function named `getLowestGammaTrail`.

2) Optimization Link: Optimization Link agents are used to create new logical links between nodes in order to optimize the topology of the overlay. The ant-like agent starts from the node requesting the connection, and migrates to its target (Figure 6). If the conditions on the hop distance between the two nodes is fulfilled (as determined by local cached information on the target node) the logical link is created and the agent migrates back to the source peer to complete its operations. The `getEstimatedDistance` function returns the estimated number of hops separating the current node from the source (according to locally cached information), whereas the `connect` function creates a new logical link to the specified node and initializes the corresponding pheromone trails.

3) Unlink: Unlink agents are used to remove logical links between nodes as result of the girth optimization rule or when a node wants to disconnect from the overlay. The Unlink agent removes the logical link information on both nodes and clears the associated pheromone trails.

4) Construction Link: In contrast to Optimization Link agents, Construction Link ones are used to initially connect new peers to the overlay as well as to recover connectivity in the event of a failure. In this regard, the logical links created by these agents have no optimization purposes.

VII. Case Study: Ozmos

As a last example we consider the implementation of a swarm-based application protocol executing on top of an existing overlay. More specifically we discuss the Ozmos protocol [45], which can be used to balance load or equally distribute resources across an overlay of nodes connected by means of a Chord overlay. In contrast to the previous examples, Ozmos agents are fully independent from overlay management agents, and their behavior is focused on relocating tasks among available resources. In contrast to other existing approaches Ozmos supports heterogeneous resources and tasks.

A. Concentration

The actual load on a node does not only depend on the number of tasks that are scheduled but also on the capability of the node itself. To determine when Ozmos should relocate jobs a normalized load value, called *concentration*, is computed by each node. The concentration of a node is proportional to its load, but is computed according to both the length of the execution schedule (i.e. total runtime), and the hardware capabilities (i.e. number of processors and relative speed) of the node. Each node receives periodic notifications about the concentration of its predecessor and successor; furthermore, a notification from a random node is also employed to improve the load balancing process. If no notification is received, the corresponding concentration value is assumed to be infinite.

B. Resources

In order to support load-balancing on heterogeneous resources the unique identifier assigned to each node by Chord is modified to embed information about the resources shared by the node. More specifically, Ozmos groups nodes into different categories depending on their hardware (i.e. architecture) and software (i.e. operating system) profile. The resource class is included in the node identifier such that all nodes belonging to the same class are adjacent on the ring. Tasks are similarly assigned unique identifiers that embed the class of resources required for their execution. Notification agents are exchanged only between nodes of the same class.

C. Agents

Ozmos employs three different types of ant-like agents in order to exchange notifications about concentration levels, to relocate tasks between nodes of the same resource class, and to relocate tasks to nodes suitable for their execution. Agents have access to local data structures, such as the local concentration, the scheduling queue, as well to information

about the underlying Chord overlay such as the address of the predecessor, the successor list, and the finger table.

1) Notification agent: Notification agents are exchanged by nodes in order to share concentration levels and information about the performance of the system (to enable normalization of the concentration values). Nodes periodically send a *Notification* agent to their predecessors, successors, and to randomly chosen nodes of the same resource class (selected from the finger table) called probes. To prevent load balancing between incompatible nodes in a heterogeneous system, notifications are not sent to nodes whose resource class is different than that of the origin node: this conditions can be easily verified, because the resource class is encoded within the node's identifier.

```
(migrate (next))

(doDetermineDirection)

(while 1 (begin
    (set! steps (- steps 1))
    (if (< steps 0) (break))
    (if (or (< (getConcentration) (nextConcentration))
            (< steps 0)) (begin
                (foreach task in $tasks
                    (schedule task))
                (end))
    else
        (migrate (next)))))
```

Fig. 7. Osmosis Agent (excerpt)

2) Osmosis: Osmosis agents perform rescheduling of the tasks in order to achieve a global load balancing of the grid system. When a node detects that neighbor peer has a lower concentration (considering a minimum threshold), it creates an *Osmosis* agent to reschedule some tasks. The behavior of the agent can be implemented using OverSwarm programming language as shown in Figure 7. An agent receives a list of tasks `$tasks` to be rescheduled from the source node, and a direction to follow in the ring (which determines, at each step, the next hop); if the agent is sent to a probe the direction is determined by following the lowest concentration after the initial migration. The agent continues traveling in the ring as long as the next peer provides a lower concentration than the current peer, until a maximum number of steps have been traveled.

3) Relocation: If incompatible tasks are submitted to a node, *Relocation* agents are created to reschedule these tasks on an appropriate resource. Figure 8 provides the basic behavior of an agent. The agent receives a list of tasks from the source node, and stores them in the `$tasks` variable. The `jump` function is then used to invoke Chord's key-based routing and migrate to a node responsible for the class of the tasks to be relocated. The key corresponds to the identifier of first node in the given resource class (i.e. a null identifier whose prefix is replaced by the class identifier).

VIII. Evaluation

In order to evaluate the performance of our framework we implemented several simple protocols using both our agent programming language, OverSim, and PeerSim. The goal of these experiments was to measure the simulation time as well as the memory consumption. All protocols are based on mobile agents that wander randomly on the network while collecting some information: although their behavior is extremely simple, the performed operations reflect common usage cases. In our simple setup we assume that each node knows each other peer in the overlay.

```
(jump (class $tasks))

(foreach task in $tasks
    (schedule task))
```

Fig. 8. Relocation Agent (excerpt)

A. Protocol A

The first protocol is based on a simple agent that wanders some random steps on the overlay collecting data. The agent starts on a node and subsequently migrates to a random peer. Subsequently, with a probability of 50% the node further migrates to a random peer, otherwise the address of the current node is stored in the variable `previous` and a migration toward a random node takes place. Finally, the result of the call to function `doSomething` is stored in variable `result`, the agent migrates back to the previous node, and either calls `doThis` or `doThat` based on the value of `result`.

```
(migrate (getRandomNode) 256)
(if (< (rand) 0.5) (begin
    (var previous (getThisNode))
    (migrate (getRandomNode) 256)
    (var result (doSomething))
    (migrate previous 256)
    (if (> result 0)
        (doThis)
    else
        (doThat)))
else
    (migrate (getRandomNode) 256))
```

Fig. 9. Example protocol A

Figure 9 represents the behavior of the agent as implemented using OverSwarm. The `migrate` function is used by the agent to migrate to another peer: its execution is hence suspended and is resumed on the target node. The `getRandomNode` function returns the address of a random node in the overlay, whereas the `getThisNode` function is used to obtain the address of the current node. The corresponding C++ methods that map to these functions are listed in Figure 10: the `blob` function is used to serialize OverSim types into a byte stream that can be transported by the agent. The additional numeric parameter 256 passed to the migration function defines the size of the payload, and is used by the underlay model to reproduce accurate transmission delays.

B. Protocol B

The second protocol employs an agent with two states: *forward* and *backward*. In the forward state the agent wanders randomly on the network, and the address of each visited peer is stored in a vector that is carried by the ant. After

```
OvSwValue::Ptr ProtocolA::getRandomNode
               (void* owner, OvSwStack* st)
{
    TransportAddress* t =
              globalNodeList->getRandomAliveNode();
    return blob(*t);
}

OvSwValue::Ptr ProtocolA::getThisNode
               (void* owner, OvSwStack* st)
{
    TransportAddress* t = &thisNode;
    return blob(*t);
}
```

Fig. 10. Example protocol A, C++ methods

a predefined number of hops, the state of the agent switches to backward: the agent goes back on its steps by migrating toward the nodes stored in its vector. Figure 11 lists the agent code as implemented using OverSwarm, whereas 12 details the message dispatch method as implemented with OverSim. Whereas in the latter the two states of the agent must be explicitly defined, with OverSwarm the behavior of the agent remains linear.

```
(migrate (getRandomNode) 256)
(var visitedNodes [])
(var remainingHops 5)
(while (> remainingHops 0) (begin
    (push visitedNodes (getThisNode))
    (set! remainingHops (- remainingHops 1))
    (migrate (getRandomNode) 256)))

(while (not (empty? visitedNodes)) (begin
    (migrate (pop visitedNodes) 256)))
```

Fig. 11. Example protocol B (OverSwarm)

```
void ProtocolB::handleProtocolMessage
                (ProtocolBMessage* msg)
{
    switch(msg->getState()) {
    case 0:
        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(*getRandomNode(), msg);
            break;
        }
    case 1:
        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. 12. Example protocol B (OverSim)

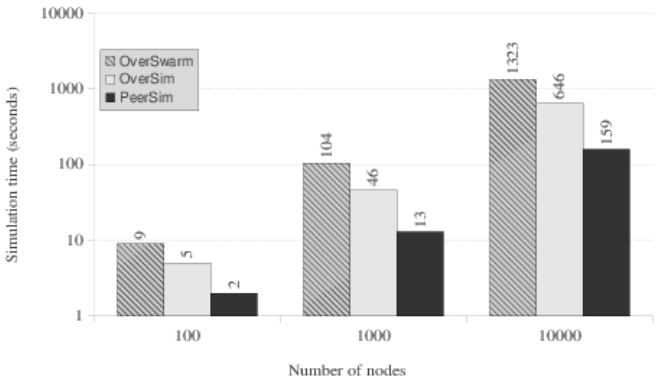

Fig. 13. Example Protocol A - Simulation Time

C. Evaluation methodology

For the evaluation overlays of different sizes have been considered: 100, 1000, and 10000 nodes. Each simulation run reproduces 6 hours of network activity; for simplicity a fully connected overlay where each node knows all other nodes was considered. Experiments were 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. Every 2 seconds each node deploys an agent which then behaves according to its species. For both OverSim and OverSwarm an underlay model based on Internet latency measurements was employed: the average latencies between peers is about 100ms. With PeerSim a simple underlay model with random latency between 50ms and 150ms was considered. The accuracy of the overlay model in OverSwarm/OverSim is higher than in PeerSim, as the latter does not consider queueing nor bandwidth effects.

D. Results

In both protocols, PeerSim benefits from its simple underlay model, and is the fastest among the three simulators, running 2.5 to 4.7 times, and 7 to 12 time faster than OverSim in Protocol A (Figure 13), respectively Protocol B (Figure 15). With both OverSim and OverSwarm the performance is affected by the additional underlay characteristics that are simulated, namely jitter and queuing effects. The additional abstraction layer introduced by our bio-inspired framework to achieve transparent migration further reduces the performance by approximately a factor of 2.3 in the test setup for protocol A, and 2.8 for protocol B; still our framework enables faster than realtime simulations, with a speed-up factor of 4 in the 10000 node experiment for protocol B. Moreover, by comparing the results obtained with the two considered protocols, we notice that the performance penalty factor is neither heavily influenced by the behavior of the agent nor by the scale of the simulated system.

Regarding memory (Figures 14 and 16), PeerSim shows a more parsimonious usage, thanks to its simplified simulation and underlay models. In particular, PeerSim message

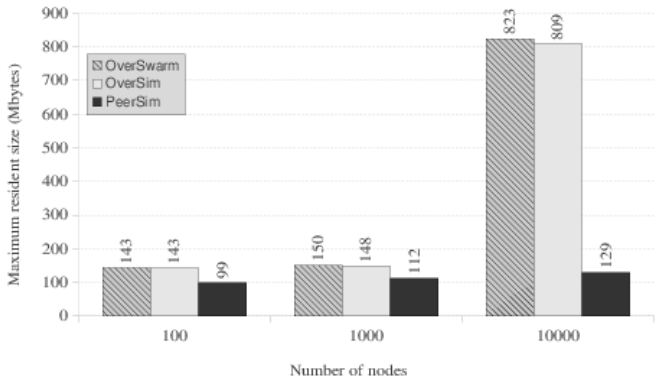

Fig. 14. Example Protocol A - Memory Usage

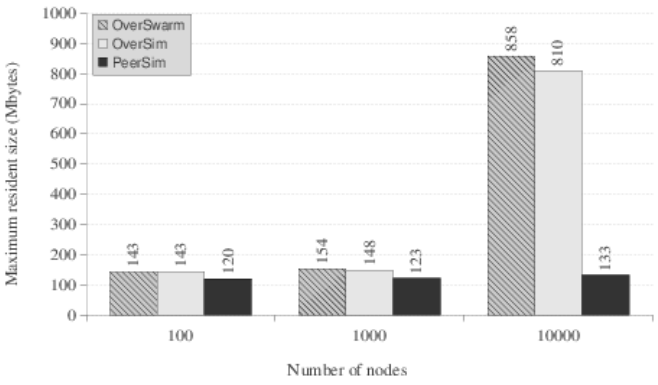

Fig. 16. Example Protocol B - Memory Usage

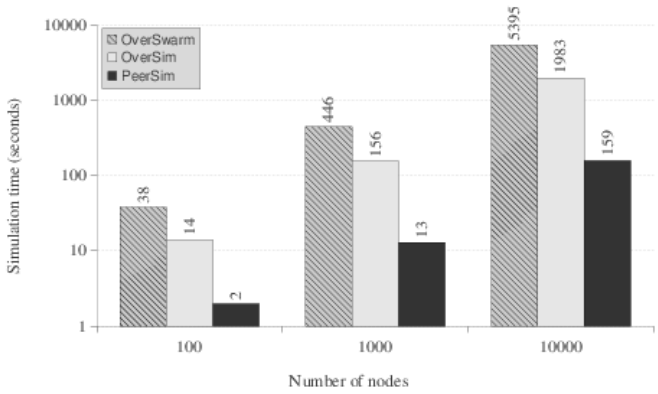

Fig. 15. Example Protocol B - Simulation Time

transmission only involves passing a reference to Java object between nodes, whereas in OverSim packet encapsulation is required to achieve accurate simulation of the underlying network. OverSwarm consumes a similar amount of memory as OverSim, meaning that the agent abstraction has little impact on this measure. These benchmark show that the price to be paid for an accurate simulation model and for bio-inspired features represents a sensible overhead, nonetheless large simulations with faster-than realtime performance are still possible. Moreover, concerning the OverSwarm platform, we argue that the overhead is also outweighed by several benefits such as strong transparent migration support and a high-level agent description language. Both these features simplify the development and readability of ant-inspired protocols and enable faster prototyping.

IX. Conclusions

In this paper we presented OverSwarm, a framework and simulation platform specifically designed for the development and evaluation of swarm-intelligence peer-to-peer protocols built on the ant-colony paradigm. These protocols are typically based on mobile agents that mimic the behavior of real insects, for example in order to determine best paths in the network. To facilitate their implementation a domain specific language that supports transparent strong migration is proposed. OverSwarm promotes a clear separation of concerns between services implemented by each peer ant the behavior of agents, and integrates with the OverSim platform to enable accurate simulation of peer-to-peer overlays with different types of underlay networks. In contrast to existing frameworks for the simulation of bio-inspired distributed systems, OverSwarm exploits OverSim's features to provide a detailed packet-level simulation, which replicates network phenomenons such as churn, queueing effects, and jitter. Our solution can be used to develop new overlay protocols, hybrid protocols that extend traditional systems with bio-inspired features, as well as application protocols that execute on top of an existing overlay. The implementation already includes some examples of bio-inspired protocols, and provides an API implementing pheromone trails. Existing protocols implemented by OverSim, such as Chord, can be easily compared with bio-inspired analogues. Current research focuses on developing a rich set of distributed algorithms based on the ant colony paradigm, and a comprehensive comparison between traditional and bio-inspired solutions. Further integration with OverSim is planned as future work to better support evaluation of both structured and unstructured overlays. Furthermore, because the code generated by the compiler does not link directly to OverSim, an adaptation of our framework to work with other simulation platforms such as MIXIM or INETMANET is envisioned. OverSwarm is actively developed and released under an open source license, and can be downloaded from http://syscall.org/doku.php/overswarm.

X. Acknowledgments

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

References

[1] Carlos Gershenson, Francis Heylighen, and Centrum Leo Apostel. When can we call a system self-organizing. In *In Advances in Artificial Life, 7th*

European Conference, ECAL 2003 LNAI 2801, pages 606–614. Springer-Verlag, 2003.

[2] Francis Heylighen. The science of self-organization and adaptivity. In *In The Encyclopedia of Life Support Systems (EOLSS), Knowledge Management, Organizational Intelligence and Learning, and Complexity. Developed under the Auspices of the UNESCO, Eolss Publishers*, 2003.

[3] Stuart A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, USA, 1 edition, June 1993.

[4] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[5] Marco Dorigo, Mauro Birattari, Thomas Stützle, Université Libre, De Bruxelles, and Av F. D. Roosevelt. Ant colony optimization artificial ants as a computational intelligence technique. *IEEE Comput. Intell. Mag*, 1:28–39, 2006.

[6] Antonio Carzaniga, Gian Pietro Picco, and Giovanni Vigna. Designing distributed applications with mobile code paradigms. In *International Conference on Software Engineering*, pages 22–32, 1997.

[7] Pengfei Di, Yaser Houri, Kendy Kutzner, and Thomas Fuhrmann. Towards comparable network simulations. Interner Bericht 2008-9, Dept. of Computer Science, Universität Karlsruhe (TH), aug 2008.

[8] Scalable Network Technologies QualNet Network Simulator. http://www.scalable-networks.com/.

[9] Ozalp Babaoglu, Hein Meling, and Alberto Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *International Conference on Distributed Computing Systems (ICDCS)*, 2002.

[10] Márk Jelasity, Alberto Montresor, Gian Paolo Jesi, and Spyros Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[11] Andras Varga. The omnet++ discrete event simulation system. *Proceedings of the European Simulation Multiconference (ESM'2001)*, June 2001.

[12] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *Proceedings of the twelfth workshop on Parallel and distributed simulation*, PADS '98, Washington, DC, USA, 1998. IEEE Computer Society.

[13] Kazuyuki Shudo, Yoshio Tanaka, and Satoshi Sekiguchi. Overlay Weaver: An overlay construction toolkit. *Comput. Commun.*, 31(2), 2008.

[14] Ion Stoica, Robert Morris, David Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings SIGCOMM '01*, volume 31, New York, NY, USA, October 2001. ACM Press.

[15] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Lecture Notes in Computer Science*, pages 329–350, 2001.

[16] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65, Berlin, Heidelberg, October 2002. Springer Berlin Heidelberg.

[17] Stephen Naicken, Anirban Basu, Barnaby Livingston, and Sethalat Rodhetbhai. A survey of Peer-to-Peer network simulators. *Proceedings of the 7th Annual Postgraduate Symposium (PGNet '06)*, 2006.

[18] Dengyi Zhang, Xuhui Chen, and Hongyun Yang. State of the art and challenges on peer-to-peer simulators. In *Proceedings of the 5th International Conference on Wireless communications, networking and mobile computing*, WiCOM'09, pages 3855–3858, Piscataway, NJ, USA, 2009. IEEE Press.

[19] Rupali Bhardwaj, V.S. Dixit, and Anil Kr. Upadhyay. Article: An overview on tools for peer to peer network simulation. *International Journal of Computer Applications*, 1(1):70–76, February 2010. Published By Foundation of Computer Science.

[20] The INet Framework. http://inet.omnetpp.org/.

[21] A. Köpke, M. Swigulski, K. Wessel, D. Willkomm, P. T. Klein Haneveld, T. E. V. Parker, O. W. Visser, H. S. Lichte, and S. Valentin. Simulating wireless and mobile networks in omnet++ the mixim vision. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 71:1–71:8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[22] András Varga. *JSimpleModule*.

[23] Andreas Lagemann and Jörg Nolte. Csharpsimplemodule – writing omnet++ modules with c and mono, 2008.

[24] The Network Simulator NS-2. http://www.isi.edu/nsnam/ns/.

[25] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A flexible overlay network simulation framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, May 2007.

[26] Frank Dabek, Ben Y. Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common api for structured peer-to-peer overlays. In *Peer-to-Peer Systems*, pages 33–44, 2003.

[27] Thomer M. Gil, Frans Kaashoek, Jinyang Li, Robert Morris, and Jeremy Stribling. p2psim: a simulator for peer-to-peer (p2p) protocols, 2006.

[28] Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, and Robbert van Renesse. Kelips: Building an efficient and stable p2p dht through increased memory and background overhead. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03*, 2003.

[29] Pedro Garcia, Carles Pairot, Ruben Mondejar, Jordi Pujol, Helio Tejedor, and Robert Rallo. *PlanetSim: A New Overlay Network Simulation Framework*. 2005.

[30] Aleksandra Kovacevic, Sebastian Kaune, Nicolas Liebau, Ralf Steinmetz, and Patrick Mukherjee. Benchmarking platform for peer-to-peer systems (benchmarking plattform für peer-to-peer systeme). *it - Information Technology*, 49(5):312–319, 2007.

[31] Brendon J. Wilson. *JXTA*. Pearson Education, 2002.

[32] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3):00–00, July 2003.

[33] Alberto Montresor and Hein Meling. Messor: Load-balancing through a swarm of autonomous agents. In *In Proceedings of 1st Workshop on Agent and Peer-to-Peer Systems*, pages 125–137, 2002.

[34] Amos Brocco, Béat Hirsbrunner, and Michèle Courant. Solenopsis: A framework for the development of ant algorithms. In *Swarm Intelligence Symposium*, pages 316–323. SIS, IEEE, April 2007.

[35] Ichiro Satoh. Test-bed platform for bio-inspired distributed systems. In *Proceedings of the 3rd International Conference on Bio-Inspired Models of Network, Information and Computing Sytems*, BIONETICS '08, pages 27:1–27:8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[36] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system: a toolkit for building multi-agent simulations. Technical report, 1996.

[37] Mrton Ivnyi, Laszl Gulys, Rajmund Bocsi, Vilmos Kozma, and Richard Legendi. The multi-agent simulation suite. In *Emergent Agents and Socialities: Social and Organizational Aspects of Intelligence*, Washington DC, USA, November 8-11 2007.

[38] Seth Tisue and Uri Wilensky. NetLogo: A simple environment for modeling complexity. 2004.

[39] I. Baumgart, T. Gamer, C. Hbsch, and C. Mayer. Realistic Underlays for Overlay Simulation. In *Proc. Proceedings of Workshop on OMNeT++*, March 2011.

[40] Agostino Forestiero, Carlo Mastroianni, and Michela Meo. Self-chord: A bio-inspired algorithm for structured p2p systems. In *Proceedings of the IEEE/ACM CCGRID '09*, CCGRID '09, Washington, DC, USA, 2009. IEEE Computer Society.

[41] Bittorrent website, http://www.bittorrent.com/, July 2011.

[42] Fernando Correia, Teresa Vazão, and Victor J. Lobo. Models for pheromone evaluation in ant systems for mobile ad-hoc networks. In *Proceedings of the 2009 First International Conference on Emerging Network Intelligence*, EMERGING '09, pages 85–90, Washington, DC, USA, 2009. IEEE Computer Society.

[43] Amos Brocco, Fulvio Frapolli, and Béeat Hirsbrunner. Bounded diameter overlay construction: A self organized approach. In *IEEE Swarm Intel ligence Symposium (SIS 2009)*, April 2009.

[44] Amos Brocco. Exploiting self-organization for the autonomic management of distributed systems, October 2010.

[45] Amos Brocco. ozmos: Bio-inspired load balancing in a chord-based p2p grid. In *Workshop on Bio-Inspired Algorithms for Distributed Systems, BADS/ICAC 2011*. ICAC 2011, June 2011.