libARA: A framework for simulation and testbed based studies on ant routing algorithms in wireless multi-hop networks

Michael Frey
Department of Computer Science
Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
frey@informatik.hu-berlin.de

Friedrich Große
Distributed, Embedded Systems
Institute of Computer Science
Freie Universität Berlin
Takustrasse 9
14195 Berlin, Germany
friedrich.grosse@fu-berlin.de

Mesut Günes
Distributed, Embedded Systems
Institute of Computer Science
Freie Universität Berlin
Takustrasse 9
14195 Berlin, Germany
mesut.guenes@fu-berlin.de

ABSTRACT

Routing in wireless multi-hop networks has been an active area of research over the last decade. Although, many testbeds were setup recently, the knowledge about the dynamic behavior of routing algorithms and the performance evaluation were mainly done via simulation. Simulation studies are imperative to understand important aspects of the algorithms, but are limited in certain aspects of the physical environment. Therefore, it is necessary to study routing algorithms also in a more realistic environment, in a testbed. Unfortunately, studying algorithms both in simulation and testbed raises multiple challenges. Particularly, the code base of the routing algorithm differ in both environments, which may lead to different behavior of the same routing algorithm.

In this paper we introduce `libARA`, a framework that supports simulation and testbed experiments to study ant routing algorithms on the very same code base. Thus, the logic of the routing algorithm is the same, but works on two different environments. This is important to keep the results from both environments comparable and thus to increase the insight into the behavior of the algorithm. The main goal of the framework is to investigate the scalability and adaptability of ant routing algorithms.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; C.2.1 [**Network Architecture and Design**]: Wireless communication; C.2.2 [**Network Protocols**]: Routing protocols

1. INTRODUCTION

For many years, routing in wireless multi-hop networks (WMHNs) has been an active area of research. This is mostly due to two facts. With the rise of wireless communication it became clear that routing protocols for wired networks are not applicable to their wireless counterparts. Furthermore, advantages of wireless networks such as interconnecting devices without special infrastructure in an ad-hoc manner has inspired many new areas of research. This in turn strengthened the demand for routing protocols in WMHNs which have to meet requirements which are critical for the functionality. This includes, but is not limited to routing protocols supporting mobility in mobile ad-hoc networks or energy efficiency in wireless sensor networks. While some researchers would consider routing in WMHNs a completed area, important questions remain open.

Many new areas of research propose the interconnection of millions of objects such as the Internet of Things (IoT) [1] or Smart Cities [2]. Crucial for the success of a routing protocol in these new areas is the ability to scale well with a increasing number of devices and the capability to adapt dynamically to the environment [3]. How do we study these requirements on routing protocols? In the past routing protocols for WMHNs have been studied in a distinct manner, typically using mathematical models, simulators, or testbeds. Mathematical models allow to formally proof that certain properties hold, but at the cost of a restricted perspective on a scientific problem. Simulation based studies allow to investigate challenges on scalability, but only provide a small set of abstractions of the physical environment. In contrast, testbeds operate in a physical environment and allow to study dynamic unexpected effects, but are considered to be expensive and limited in the network size. From a scientific methodology perspective the foremost question is if we can still study routing algorithms in a distinct manner.

Liotta argues in [4] that existing routing protocols will fail to meet the requirements on large scale networks and proposes to focus on bio-inspired algorithms in order to tackle the challenge on scalability and adaptivity. One prominent example for bio-inspired algorithms are swarm intelligence algorithms which are inherently self-organizing [5]. Here, individuals in a swarm behave on a local level on a simple set

ValueTools'13, December 10 – 12 2013, Turin, Italy

VALUETOOLS 2013, December 10-12, Torino, Italy

DOI 10.4108/icst.valuetools.2013.254382

of rules. This simple set of rules in interaction with other individuals of the swarm emerges to a complex pattern on a global level. This *emergent behavior* allows a swarm to solve a global problem. Examples for swarm intelligence algorithms are synchronization algorithms based on fireflies [6], bird swarms in cluster formation [7], or ant algorithms in routing [8].

We want to combine simulation and testbed based studies and investigate the scalability and adaptivity of routing algorithms. In particular, we focus in our work on ant routing algorithms. Ant routing algorithms are based on the Ant-Colony Optimization (ACO) metaheuristic [9] and are inspired by the foraging behavior of ants. Here, ants mark favorable paths towards food with a special hormone, called pheromone. Ants are attracted by pheromones and in turn reinforce them on the trail. However, pheromones evaporate over time and thus only the shortest path towards a food source remains. Ant routing algorithms have been studied extensively in simulation, particularly AntNet [10], AntHocNet [11], and ARA [12]. However, there are no performance studies on ant routing algorithms in a physical environment [8].

The contribution of this paper is `libARA`, which is a framework that supports simulation and testbed experiments to study ant routing algorithms on the very same code base. The logic of the routing algorithm is the same, but works on two different study environments. This is important to keep the results from both environments comparable and thus to increase the insight into the behavior of the algorithm. The main goal of the framework is to investigate the scalability and adaptability of ant routing algorithms.

The remainder of this paper is organized as follows. First, we introduce our approach in Section 2 and present a proof of concept implementation of two ant routing algorithms, namely ARA and its successor, the energy aware ant routing algorithm (EARA) in Section 3. We discuss related work in Section 4. Finally, we give a short summary and conclude the paper with a outlook on future work.

2. THE LIBARA FRAMEWORK

The framework is called library for ant routing algorithms (`libARA`). It provides different study environments, such as network simulators and wireless testbeds and is written in C++. `libARA` provides a set of classes, which enables a researcher to implement new ant routing algorithms in a short time. While we provide implementations for a specific simulator and a routing framework used in the DES research group, `libARA` stands for its own. The framework is highly modularized.

2.1 Architecture

Fig. 1 depicts the architecture of `libARA`. It consists of the abstract core, denoted as `libARA` in the figure, and the implementations for a discrete event based simulator and a wireless testbed. The abstract core contains classes which are independent of the respective implementations. This includes typical concepts of a routing algorithm such as the routing table, addresses, or network interfaces. In addition, there are policies which correspond to certain aspects of ant routing. As previously mentioned, the reinforcement and evaporation processes control indirectly which paths ants favor. The reinforcement and evaporation policies in the figure represent these two processes. The forwarding policy represents the transmission probability in an ant routing algorithm. While this modularization introduces a slight overhead, it allows to extend the framework in a fast way and enables researcher to provide new implementations of other ant routing algorithms easily.

For the simulation implementation we use `OMNeT++` [13] and the INETMANET. For the testbed implementation we use the DES - Simple and Extensible Routing-Framework for Testbeds (DES-SERT) [14], a framework for the implementation of proactive, reactive, and hybrid routing protocols. The framework consists of the actual library libdessert and an additional library libcli, which provides a command line interface for the management of routing daemons.

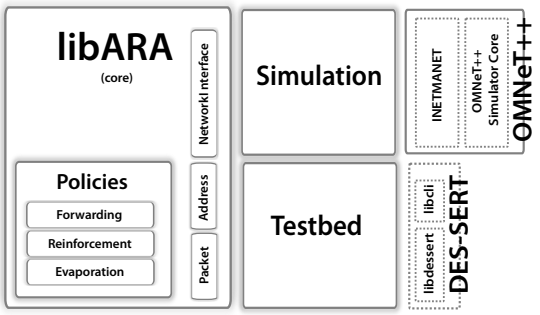

Figure 1: Architecture of `libARA` on a package-level. The framework consists of an abstract core and implementations for simulation and testbed based studies.

We provide the core routing logic in `libARA` via the `AbstractNetworkClient` class. Fig. 2 depicts the `AbstractNetworkClient` and related classes. The abstract `AbstractNetworkClient` provides methods for sending and receiving packets. Furthermore, it is also responsible for delivering packets to upper layers or notifying the next higher layer about undeliverable packets. We split the logic of this abstract class in multiple classes. A client (`AbstractNetworkClient`) uses a network interface (`NetworkInterface`) in order to communicate with its surrounding environment. If there is no entry in the routing table (`RoutingTable`) for a specific destination of a packet (`Packet`), the framework keeps the packet in a special data structure (`PacketTrap`) until the algorithm has found a route or canceled the route discovery (e.g. it could not establish a route). Since the core of the `libARA` framework is only provided by means of abstract classes, a routing daemon must implement all required virtual interfaces. This includes the addresses, the network interface and a concrete client.

2.1.1 Address

The `Address` interface represents the address of a host in a network. Typically, for routing purposes an IP address is used. However, the type of the actual address is highly dependent of the desired implementation. While we use IPv4 addresses in the simulator, we use MAC addresses in the testbed version. This is due to the fact that the routing framework performs a transparent underlay routing. The `Address` interface itself is fairly simple. It provides methods for the comparison of addresses, string representation, and hashing.

2.1.2 Packet

The foremost task of a routing daemon is to forward packets. The `Packet` class represents a packet consisting of source and destination address, a time-to-live (TTL) and a sequence number. The `PacketFactory` in turn creates instances of the class `Packet` and follows the factory pattern [15]. The framework receives and sends packets via a network interface, represented by class `NetworkInterface`.

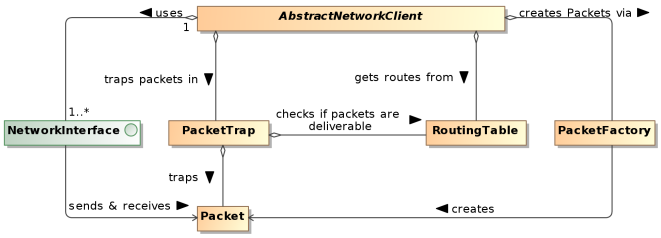

Figure 2: A class diagram of the `AbstractNetworkClient` and its closely related components.

2.1.3 Network Interface

The `NetworkInterface` interface is a abstraction of a network interface card (NIC). It provides methods for sending packets over the network and receiving packets from the network. Every `NetworkInterface` is aware of its own address and is aware if a address in packet is a broadcast address or not. We provide a basic `NetworkInterface` implementation by means of the `AbstractNetworkInterface` class. Besides the basic functionality, the class also supports multiple network interfaces. Every class which inherits from `AbstractNetworkInterface` class will only need to implement the corresponding send and comparison methods. The framework requires the latter in order to distinguish multiple network interfaces.

2.1.4 Client

A client in `libARA` encapsulates the logic of a routing algorithm. Basically, this involves two main tasks. First, a implementation has to provide a packet dispatching mechanism. A packet dispatcher receives data from the lower network layer and creates instances of class `Packet` from this information. The framework has to encapsulate the extracted data correctly in a `Packet` so a client can process the information accordingly. Second, a developer has to implement the handling for upper layers, which includes the passing of packets for this node to the upper layer or reporting packets as non deliverable if possible. Hence, the resulting implementation is relatively small. In effect, all of the custom implementations for the routing daemon only serve for the communication with the surrounding environment.

3. PROOF OF CONCEPT: ARA AND EARA

We provide a proof of concept implementation of ARA [12] and its successor EARA [16] for simulation and wireless testbeds based on `libARA`. Since the core of the `libARA` framework is only provided by means of abstract classes, an ARA or EARA routing daemon must implement all of the required virtual interfaces.

Fig. 3 shows the extended class diagram for the `AbstractARAClient` class and its related classes. The `EvaporationPolicy` and `PathReinforcementPolicy` classes represent the positive and negative feedback in ant routing algorithms. The `ForwardingPolicy` class provides an abstraction for the transmission probability in ant algorithms. The class `AbstractARAClient` inherits from the abstract `AbstractNetworkClient` class and provides the ARA routing logic. The realization of the `Address` interface defines which type

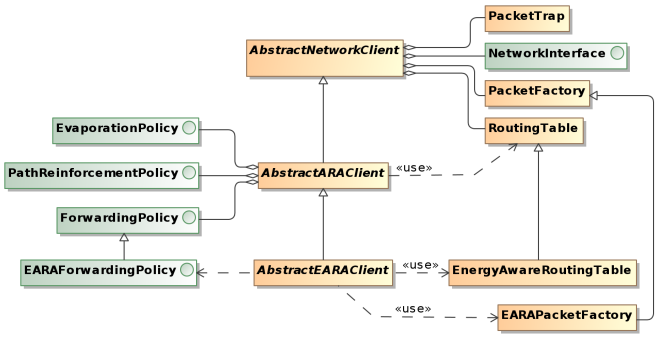

Figure 3: Extended class diagram of the `AbstractARAClient`, `AbstractEARAClient`, and related components.

of addresses a implementation uses. In the simulation we use IPv4 (`IPv4Address`) addresses provided by the `OMNeT++` INET package. Since DES-SERT provides routing by means of underlay routing we use MAC addresses in the testbed respectively. While the routing logic of EARA slightly differs to ARA, we provide a `AbstractEARAClient` class which represents the EARA client. The class also inherits from class `AbstractNetworkClient` and uses a specialized `ForwardingPolicy`.

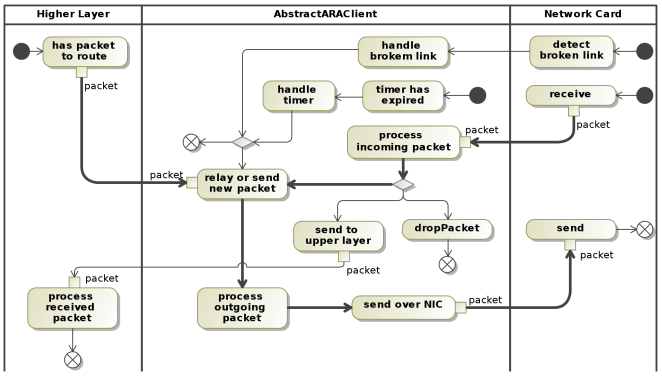

Figure 4: An activity diagram of the top level program flows. The bold arrows denote the send and relay control flows.

Fig. 4 depicts the top level program flow. The activity diagram consists of three entities: the upper layer, the `AbstractARAClient` and the network card. The routing of a packet from the upper layer will always result in a packet sent via the network card, also referred as NIC. The foremost question is if the packet can directly be routed or if a new route discovery is started. The reception of a packet can also result in sending over the NIC (relaying), but it might as well happen that the received packet is dropped

or delivered to the higher layer. The client handles expired timers or broken links and might send further packets into the network. The program flow in the `AbstractEARAClient` is similar to the `AbstractARAClient` and thus omitted.

3.1 Packet

While `libARA` provides a generic `Packet` implementation, the simulation and testbed implementations require a specialized version. `OMNeT++` provides its own message definition format and a generator for creating classes related to messaging and packets. In general, it is feasible to implement messaging and packet primitives in `OMNeT++` without using the built-in facilities, but it is not recommended at all. DES-SERT provides a data structure for routing daemon specific data, payload and tracing extensions for debugging purposes. The data structure in turn is added in the routing framework as payload to ethernet frames before it is actually sent. Both, simulation and testbed, use the existing packet data structures and classes in the respective frameworks. This requires a specialized `PacketFactory` and customized methods for the classification of messages.

3.2 NetworkInterface

The `ReliableNetworkInterface` class implements a basic network interface. The class extends the `AbstractNetworkInterface` by means of acknowledgment mechanisms. The implementation for the simulator inherits from this class and adds various statistics related methods which are simulation-specific. While one would expect a distinct implementation for the testbed which interacts directly with the NIC, we rely solely on the DES-SERT library and reuse most of the methods defined in the `ReliableNetworkInterface`.

3.3 Time

We provide a framework within `libARA` for the concept of time. Particularly, we provide classes and methods which allow to store and compare timestamps and do calculations on them. In Fig. 5 we present an overview of the time related classes. We omit classes in the figure which are related to the testbed for the sake of clarity. The time framework is

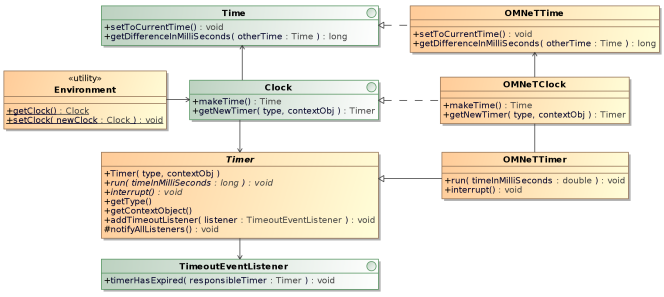

Figure 5: The class diagram for `Environment`, `Clock`, `Time`, `Timer`, `TimeoutEventListener`, and related classes.

mostly guided by a factory and singleton pattern [15]. The core of the time framework is the `Clock` interface, which acts as a factory for instances of class `Time` and `Timer`. Whenever the client or another component needs a new timer or some notion of time, it can access this clock using the static `Environment` class. The idea is that every single module in `libARA` can easily access the clock via this class without having to know about the existence of the clock in the first place. The time framework consists of purely virtual interfaces or abstract classes. We provide implementations for the simulation (`OMNeTClock`, `-Time`, and `-Timer`) and the testbed (`StandardClock`, `-Time`, and `-Timer`) part of `libARA`. The implementation of the time framework for the testbed uses the system time in order to determine a point in time and compute time differences. Here, we rely on the `std::chronos` and `std::duration` classes introduced in C++11 [17]. Following the abstract approach of the timer framework, the framework uses the `StandardClock` in order to create instances of `StandardTimer` and `StandardTime`. The `StandardTimer` represents an independent sleeping thread in the context of the operating system, which wakes up and notify its listeners if a given timer interval is exceeded.

3.4 Client

`libARA` provides implementations for the ARA (`AbstractARAClient`) and the EARA (`AbstractEARAClient`). Both classes provide the core routing logic in the framework. However, every concrete implementation has to implement the `deliverToSystem()` and `packetNotDeliverable()` methods. Both methods handle the communication with the next higher layer and have to handle the passing of packets for this node to the upper layer or reporting packets as non deliverable if possible. In addition, we implemented a packet dispatching mechanism, which handles the reception of data from the next lower network layer and creating an instance of class `Packet` from that information. All relayed information is correctly encapsulated in this packet so the `AbstractARAClient`, respectively `AbstractEARAClient`, can process the information accordingly. Hence, the resulting concrete ARA implementation is relatively short and all of the routing logic is part of the core of the framework. In effect, all of the custom implementations for the routing daemon only serve for the communication with the surrounding environment. Simulation and testbed provide a implementation of its own client which inherits from `AbstractARAClient` or `AbstractEARAClient` respectively. In contrast to the testbed part, the client for the simulation adds additional functionality for initialization and statistics which follows `OMNeT++` requirements. The design of the related classes for the simulation also follows the design principles of routing protocols in `OMNeT++`'s INETMANET package. The testbed follows a different design approach. Here, DES-SERT assumes that a developer splits the different tasks of a routing daemon in separate, prioritized callback functions. The developer registers the priority in combination with a function pointer of the callback function in DES-SERT. For every packet DES-SERT calls the registered functions, starting with the function with the highest priority. Thus, DES-SERT realizes a message pipeline. Since we implemented `libARA` in C++ and it is not possible to pass method signatures to the callback registration function in DES-SERT, we had to find a different approach. We introduced a class `PacketDispatcher` which provides two callback methods for DES-SERT. The first callback reads data from a network interface, while the second callback writes data to it. Hence, this allows us to use most of the existing functionality of `libARA`.

4. RELATED WORK

There are several approaches to study swarm intelligence

algorithms and each one focuses on a small set of research questions. Typically, researchers study the behavior and dynamics of swarm intelligence algorithms using multi-agent systems. One example for such a multi-agent system is the Swarm Simulation System [18]. It focuses on the evaluation of coordination mechanisms within swarms. Here, a swarm represents a collection of agents executing specific scheduled actions. The platform allows to create hierarchical and nested models, consisting of agents from different swarms. The Multi-Agent Simulation Suite (MASS) [19] provides an environment for the development, simulation and evaluation of agent-based systems. Since many researchers in the area of complex systems are not necessarily computer scientists, the tool suite allows users with limited programming skills to create complex multi-agent systems on the basis of program assistants. NetLogo [20] is a multi-agent programmable modeling environment and allows to experiment visually with complex systems. It provides an extensive library, consisting among others, of models of biology, physics, and networks. NetLogo has been widely used among students, teachers and researchers for modeling and understanding dynamics of complex systems. However, agent-based approaches do not provide abstractions of a concrete application domain. In particular, models common in wireless communication are not supported. Thus, frameworks combining swarm algorithms and network simulation have evolved over the past few years. Many of these systems focus on the application of ant colonies in higher layer applications in terms of the ISO/OSI model. One example for such a system is AntHill [21]. The authors developed AntHill in order to to support the development, evaluation and deployment of bio-inspired peer-to-peer (P2P) applications. AntHill provides a cycle-based simulation of protocols and a distributed deployment using JXTA [22], an open source P2P protocol specification. Like our approach, AntHill aims at studying the behavior of ant algorithms in simulation in a large scale. In contrast to our approach, implementations in AntHill do not share the same data structures for simulation and testbed. Furthermore, AntHill's simulation part does not allow to study the networks behavior. The authors discarded the development of the project in 2002 in favor of the development of the more generic PeerSim [23] simulator.

Solenopsis [24] provides a framework for the development of ant algorithms for network management tasks. It consists of a Lisp like domain specific language (DSL) for the specification of ant agents and an execution environment. Since the focus of the framework is network management, it provides methods for specifying ant agents requirements in terms of computational resources. In addition, Solenopsis employs a simple event-based simulator which allows to study the latency among other metrics. Further studies on problems and challenges in communication are not supported.

Overswarm [25] is a framework for the development and evaluation of ant colony based algorithms for P2P overlays. It uses the OverSim [26] platform, an extension for `OMNeT++` and provides a DSL for the definition of agents behavior. The DSL generates C++ code out of the agents defined behavior, which in turn allows it to integrate it into the OverSim platform. The framework itself does not support studies in wireless testbeds, but uses the emulation mode of OverSim. The authors use for emulation the underlay support of `OMNeT++`'s INET project.

All presented approaches, except Overswarm, strictly tie the modeling of the algorithm and the underlying technology together. Overswarm relies on OverSim which in turn is an extension of `OMNeT++`. This allows to study algorithms developed in Overswarm and other existing algorithms in `OMNeT++` together. However, there are no approaches for swarm algorithms which combine simulation and testbed studies. An approach towards such a solution is the Click Modular Router (Click) [27]. Click is a framework for the test and development of communication protocols written in C++. It is available in different variants including a userspace program, kernel modules and network simulators. The main concept of Click are *elements* which provide functionality such as classification, scheduling, queuing or interaction with network devices. The *push*/*pull* connections interconnect elements and specify how data flows between elements. While developers can run code written for Click both in simulation and on actual physical hardware, this seems to be a exception rather than the rule. Despite the fact that there are no elements on ant routing available in Clicks package repository, many of these elements might be helpful for analysis and evaluation of routing protocols. However, there is no framework which allows to model a swarm-intelligence algorithm and transform this description to Click.

5. CONCLUSIONS

Routing in WMHNs has been an active area of research over the last decade. Typically, researchers study routing in simulators or testbeds. While simulations allow to capture certain aspects of routing algorithms, they are limited in terms of abstractions of the physical environment. In contrast, testbeds allow to capture observations of the physical environment, but are considered expensive and limited in network size. However, new areas of research require to study the scalability and adaptability of routing algorithms. While scalability issues are best studied in simulation, testbeds allow to study the dynamic effects of wireless communication. In this paper we presented `libARA`, a framework for the combined study of ant routing algorithms in simulation and testbed. It shares its code base among simulation and testbed. Thus, the logic of the routing algorithm is the same, but works on two different environments. The framework provides a proof of concept implementation for two approaches, ARA and EARA. Our current research focuses on conducting initial studies using our methodology and `libARA`. Future work will include the development of variants of ant routing algorithms for `libARA` and the investigation of performance. This will allow us to compare our algorithms to other state of the art ant algorithms and to optimize ARA and EARA in respect to metrics such as the network lifetime, delay, or throughput. A further interesting approach might be to follow approaches such as in systems such as Solenopsis or Overswarm and to provide a DSL for specifying ant routing algorithms or at least parts of it. While the framework itself is highly modularized and provides abstractions of core concepts of routing algorithms, it seems feasible to include other bio-inspired networking algorithms which allow us to study the performance of these systems in respect to scalability, adaptability, and self-organization in general. We certainly hope that our framework will encourage other researchers in bio-inspired networking and in particular ant routing to provide new implementations of their respective algorithms. A wide selec-

tion of ant routing algorithms would ease the evaluation of both, state of the art and new ant algorithms, and bring forward the research in bio-inspired networking. `libARA` is under active development and released under an open source license. The latest release and the source code is available at http://www.github.com/des-testbed/libara.

6. REFERENCES

[1] Kevin Ashton. That 'Internet of Things' Thing. *RFiD Journal*, 22:97–114, 2009.

[2] Joachim Fischer, Jens-Peter Redlich, Björn Scheuermann, Jochen Schiller, Mesut Günes, Kai Nagel, Peter Wagner, Markus Scheidgen, Anatolij Zubow, Ingmar Eveslage, Rober Sombrutzki, and Felix Juraschek. *From Earthquake Detection to Traffic Surveillance–About Information and Communication Infrastructures for Smart Cities*, page 121–141. Springer, 2013.

[3] Frey, Michael and Günes, Mesut. Combining Control Loops and Ant Algorithms to Optimize Network Lifetime. In *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM), Poster Session*. IEEE, 2013.

[4] Antonio Liotta. The Cognitive Net Is Coming. *IEEE Spectrum*, August 2013.

[5] Falko Dressler. *Self-Organization in Sensor and Actor Networks*. John Wiley & Sons, 2007.

[6] Ozalp Babaoglu, Toni Binci, Márk Jelasity, and Alberto Montresor. Firefly-inspired Heartbeat Synchronization in Overlay Networks. In *Self-Adaptive and Self-Organizing Systems, 2007. SASO'07. First International Conference on*, pages 77–86. IEEE, 2007.

[7] SM Guru, SK Halgamuge, and S Fernando. Particle swarm optimisers for cluster formation in wireless sensor networks. In *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. Proceedings of the 2005 International Conference on*, pages 319–324. IEEE, 2005.

[8] Laurent Paquereau and Bjarne E Helvik. Ant-Based Systems for Wireless Networks: Retrospect and Prospects. In *Self-Organizing Systems*, pages 13–23. Springer, 2012.

[9] Marco Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[10] G. A. Di Caro. *Ant Colony Optimization and its application to adaptive routing in telecommunication networks*. PhD thesis, Faculté des Sciences Appliquées, Université Libre de Bruxelles, Brussels, Belgium, November 2004.

[11] Frederick Ducatelle. *Adaptive Routing in Ad Hoc Wireless Multi-Hop Networks*. PhD thesis, Universita della Svizzera italiana, Lugano, Switzerland, 2007.

[12] Mesut Günes, Udo Sorges, and Imed Bouazizi. ARA-the ant-colony based routing algorithm for MANETs. In *Parallel Processing Workshops, 2002. Proceedings. International Conference on*, pages 79–85. IEEE, 2002.

[13] András Varga et al. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)*, volume 9, page 185, 2001.

[14] Bastian Blywis, Mesut Günes, Felix Juraschek, Philipp Schmidt, and Pardeep Kumar. DES-SERT: A Framework for Structured Routing Protocol Implementation. In *IFIP Wireless Days 2009*, Paris, France, 2009.

[15] Robert Cecil Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2003.

[16] Friedrich Grosse. Optimization and Evaluation of Energy Aware Ant Routing Algorithm Strategies based on Network Simulations. Master's thesis, Department of Computer Science, Freie Universität Berlin, Germany, August 2013.

[17] International Standard ISO/IEC 14882:2011(E) – Programming Language C++.

[18] Nelson Minar, Roger Burkhart, Chris Langton, and Manor Askenazi. The swarm simulation system: A toolkit for building multi-agent simulations. Technical report, 1996.

[19] Marton Ivanyi, Rajmund Bocsi, Laszlo Gulyas, Vilmos Kozma, and Richard Legendi. The multi-agent simulation suite. In *Emergent Agents and Socialities: Social and Organizational Aspects of Intelligence. Papers from the 2007 AAAI Fall Symposium*, pages 57–64, 2007.

[20] Seth Tisue and Uri Wilensky. NetLogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, pages 16–21, 2004.

[21] Ozalp Babaoglu, Hein Meling, and Alberto Montresor. Anthill: A framework for the development of agent-based peer-to-peer systems. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, pages 15–22. IEEE, 2002.

[22] Li Gong. JXTA: A network programming environment. *Internet Computing, IEEE*, 5(3):88–95, 2001.

[23] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, 2009.

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

[25] Amos Brocco and Ingmar Baumgart. A Framework for a Comprehensive Evaluation of Ant-Inspired Peer-to-Peer Protocols. In *Parallel, Distributed and Network-Based Processing (PDP), 2012 20th Euromicro International Conference on*, pages 303–310. IEEE, 2012.

[26] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A scalable and flexible overlay framework for simulation and real network applications. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*, pages 87–88. IEEE, 2009.

[27] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.