Policy-Based management of Overlay Networks
Master Thesis


For the degree of: Master of Science in Computer Science: Networks. University of Bern.

Switzerland. 17/09/2010

AUTHOR
Jorge Peña Cotarelo

Supervisors
Dr. Apostolos Malatras
Dr. Michèle Courant
Prof. Béat Hirsbrunner
Preface

During the last months I have lived in the city of Fribourg (Switzerland) for the development of my Master Thesis. I have enjoyed the time spent in the pleasant, challenging and innovative Pervasive and Artificial Intelligence research group at the department of Informatics in the University of Fribourg. Thanks to this work, and besides learning about the topics related with this thesis, I also learned effective methods for research, the academic writing, and the standards of Swiss universities for theses.

Firstly I want to thank to my supervisor Dr. Apostolos Malatras for his continual guidance, encouragement and for providing me feedback on my work that motivated me to learn from my mistakes. Secondly, I would like to thank the professors Dr. Michèle Courant and Prof. Béat Hirsbrunner for their feedback and especially for trusting me in the first stage. I also thank to all PhDs, Postdocs and all the staff of the PAI and the Informatics department at the University of Fribourg, because they were always friendly and helpful when I asked for support.

Finally, I am thanking my family for their moral and economical support, my friends especially those who I met in Switzerland, and last but not least I thank my girlfriend Olga for her everyday support, her encouragement in the good and bad moments, and for supporting my idea of having higher education even when a thousand of kilometers were between us.


“Everybody is ignorant, only on different subjects”
Will Rogers
# Table of Contents

1. Introduction ........................................................................................................................ 7
   1.1. Context ........................................................................................................................ 7
   1.2. Motivation ................................................................................................................... 9
   1.3. Goal .............................................................................................................................. 9
   1.4. Methodology ............................................................................................................. 10
   1.5. Structure of the document ........................................................................................ 12

2. State of the art ................................................................................................................. 13
   2.1. Overlay networks ....................................................................................................... 13
      2.1.1 Overview ............................................................................................................. 13
      2.1.2 BlatAnt ................................................................................................................ 14
   2.2. Network management ............................................................................................... 16
      2.2.1 Overview ............................................................................................................. 16
      2.2.2 Policy-Based Network Management .................................................................. 17
   2.3. Policy-Based Network Management related work ................................................... 20

3. Sensitivity analysis of BlatAnt........................................................................................... 25
   3.1. Comparison of performance in different topologies ................................................ 25
   3.2. Comparison of parameters ....................................................................................... 26
      3.2.1 Alpha Table Maximum Size ................................................................................ 26
      3.2.2 Ant Vector Length .............................................................................................. 28
      3.2.3 Optimization parameter D ................................................................................. 30
      3.2.4 Birth Probability ................................................................................................. 32
   3.3. Summary of the sensitivity analysis .......................................................................... 34
   3.4. Scenarios and policies ................................................................................................ 34
   3.5. Modification of parameters during runtime ............................................................. 35
      3.5.1 Alpha table maximum size ................................................................................. 36
      3.5.2 Ant vector length .............................................................................................. 37
      3.5.3 D parameter ....................................................................................................... 38
      3.5.4 Birth Probability ................................................................................................. 39
      3.5.5 Discussion ........................................................................................................... 40

4. Design of a Policy-Based overlay network management system .................................... 41
   4.1. Requirements ............................................................................................................ 41
      4.1.1 Scope .................................................................................................................. 41
      4.1.2 Conceptual structure .......................................................................................... 41
      4.1.3 Typical scenario .................................................................................................. 42
4.2. System design and software development ............................................................... 43
  4.2.1 System architecture ........................................................................................... 43
  4.2.2 Flowchart .......................................................................................................... 45
4.3. Development tools and techniques ........................................................................ 46
  4.3.1 Multi-threading .................................................................................................. 47
  4.3.2 Sockets ................................................................................................................ 47
  4.3.3 Database ............................................................................................................. 47
  4.3.4 Database connector ........................................................................................... 48
  4.3.5 BlatAnt ................................................................................................................ 48
4.4. Software elements..................................................................................................... 50
  4.4.1 Context ............................................................................................................... 50
  4.4.2 PDP ..................................................................................................................... 50
  4.4.3 Policy Editor ........................................................................................................ 51
  4.4.4 PolAnt ................................................................................................................. 53
5. Policy-Based management of the BlatAnt overlay network ............................................ 58
  5.1. Overview .................................................................................................................... 58
  5.2. Default policies .......................................................................................................... 58
  5.3. Analysis of the Policy-Based experiments ................................................................ 59
  5.4. Single policy applied .................................................................................................. 60
    5.4.1 Policy: Regeneration ........................................................................................... 60
    5.4.2 Policy: Reduce connections ................................................................................ 61
    5.4.3 Policy: Stability ................................................................................................... 62
    5.4.4 Policy: High Robustness .................................................................................... 64
  5.5. Multiple policies ........................................................................................................ 65
    5.5.1 Robustness & Stability ........................................................................................ 66
    5.5.2 Robustness & Reduce connections .................................................................... 67
  5.6. Discussion .................................................................................................................. 68
6. Conclusion ........................................................................................................................ 70
7. References ........................................................................................................................ 72
8. Annex ................................................................................................................................ 76
  8.1. Source code (CD-ROM) .......................................................................................... 76
  8.2. Database Information (CD-ROM) ............................................................................. 76
  8.3. Complete set of result graphs for the sensitivity analysis........................................... 76
Abstract

Overlay networks provide advantages for a better utilization of the increasingly growing number of network resources. However, often the overlay networks need to deal with dynamic changes and volatility of resources. Such situations can decrease the performance of the network when they are not explicitly treated by the overlay algorithm. Therefore, overlay networks need a global management scheme to reach and maintain an efficient operation. Policy-based overlay network management is a promising approach that can bring automatic reconfiguration and hence self-autonomy to the management of the network overlays. In this Master Thesis, a prototype of a Policy-based Overlay Network Management System capable of optimizing a bio-inspired overlay network according to changes in the network context is presented. The designed solution brings a semi-autonomic maintenance of the overlay network, easing drastically the work of the network administrator. Results obtained from a simulation analysis of the proposed approach indicate the effectiveness of our solution and are also encouraging in terms of efficiency.

Keywords: Policy-based management, overlay network management, autonomic management, overlay network maintenance, context awareness.
1. Introduction

1.1. Context

Providing high computing power to everybody at any moment is a challenge for what is often called “Internet of Computing” [1]. The Pervasive and Artificial research group of the University of Fribourg is contributing to the idea of distributed computing by constructing a model algorithm to enable monitoring and scheduling on grid network with the aim of bringing an increase in Quality of Service, robustness and efficiency for applications in grid computing. Grid computing facilitates the combination of computer resources from multiple administrative domains towards a common goal [2]. The “Internet of Computing” can provide a higher performance in computing, communication or storage than what is achieved on a single dedicated machine.

Grid computing has been successfully used in scientific applications like SETI@home [3] in which different complex tasks are divided in several parts and are sent to the different nodes that compose the grid network to achieve a common goal, namely to compute collectively an overall complex task. Each part is sent to a number of nodes on the network in order to ensure robustness in case of failures and every node computes its task independently. In a distributed scenario, each node is responsible for the distribution and the execution of the received task, like resource discovery, availability, routing, etc., to finally collect and join the results. The whole process cannot be launched if there are not enough resources in the network or if there are not enough nodes to perform the task.

However, the resources on a grid network can be heterogeneous (different architectures like supercomputers, clusters, etc.) and additionally, such resources could be connected through heterogeneous and wide area networks (i.e. internet). Because of the existence of several administrative domains, of the heterogeneity and volatility of resources, there is a need for transparency to automatically and efficiently organize and schedule tasks. Overlay networks can be considered as one of the possible mediums for organizing and coordinating these tasks. An overlay network is a virtual computer network which is built on top of a real network [4]. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which could consist of a path of possibly many physical links in the underlying network.

Overlay networks can be used to manage the complexity of highly heterogeneous and complex networks. By overlay network management we refer mainly to the creation, optimization and termination of routes within the overlay network. The recent trend of mobility in the communications [5] is reflected by the research done by the PAI group, and such research is evolving to focus on the study of the overlay networks in a mobile scenario [6]. In mobile scenarios, the frequent volatility of resources or the mobility of the overlay nodes, can affect the performance of the overlay network. Therefore, the management of overlay networks requires constant adaptation.

Policy is a common terminology when thinking in regards to administrating large computer systems. Normally, in large scale networks there are simultaneously multiple events that (if no measures for management of the network are implemented) could lead to a reduction in
the performance can be reduced such as congestion, timeouts, overloading of links because of routing rules, etc. In the Policy-Based Network Management systems, policies are defined as rules that govern the states and behaviors of a computer network. “Policies are used to specify what actions must or must not be performed by the environment on the occurrence of certain events” [7]. Typically policies are Event-Condition Action (ECA) rules. These rules are defined to assure the performance, reliability and safety of the network. Potentially, such rules could be adjusted automatically by monitoring the context, in order to balance the overlay network depending on the requirements of the grid.

With Policy-based management for overlay networks, long term and network wide configuration rules can be specified for adapting to new application requirements or evolutionary changes in the overlay network. When a change of context occurs in the underlying network, the overlay network structure is reconfigured automatically according to the high level objectives defined previously by the network administrator, to optimize the underlying structure according to the context information.

The key point of a Policy-Based Overlay approach is the separation of the rules that adapt the behaviour of the overlay network to the actual operation of the overlay network, thus creating a flexible and extensible solution. Hence in Policy-based network management we are able to adapt the behaviour of the overlay network, without the need to recode the operation of the overlay network algorithm, avoiding therefore the implementation of a new algorithm that adapts to the context of the underlying network.

In this thesis, the current state of the art supports the concepts developed, and also, it discusses improvements related with the completely distributed nature of a Policy-based system by the learning from the context to adapt the policies, among others. The work realized in this Master Thesis contributes to the current state of the art in network management proving that when overlay network is affected by changes in the context that affects the network performance, Policy-based Overlay network management can be successfully applied to reconfigure the overlay network structure for an optimal operation. The main goal of this thesis was to develop a Policy-Based Overlay network management system that bring an autonomic reconfiguration of an overlay network by means of policies, when changes of context are detected, in order to optimize the overlay operation of the overlay algorithm.

An important point is that the different steps used in this research could be applied for the evaluation of a different algorithm of overlay network creation with a Policy-based management. To validate the correctness of our statements, a policy based management solution was designed for autonomic optimization of an overlay network and validated with success. We built our solution based on the work presented in [8] [9] which uses a bio-inspired approach to construct an overlay network based on how ants discover and construct paths towards a food source and how they optimize the routes for the collection of such resources when they are out of the nest.
1.2. Motivation

This project is targeted at the management of pervasive environments, to support distributed computing by constructing a model algorithm to enable monitoring and scheduling on the SmartGRID project [9]. The Pervasive Artificial Intelligence (PAI) group from the University of Fribourg is in the process of developing a network overlay for these environments. The BioMPE research project addresses adaptive and reliable communication services based on P2P overlay networks that are built on top of highly heterogeneous, dynamically changing, wired and wireless network infrastructures. One of the goals of this bio-inspired overlay network is to hide the complexity and instability of the underlying network by offering reliable services based on distributed ant algorithms. The distributed ant algorithms support building and maintaining the overlay network. In this respect, a simulator that will allow for the evaluation of the mechanisms developed in BioMPE will be built. One important aspect that needs to be considered in these settings is the support for the dynamicity and the evolution of the system based on monitored conditions. The initial configuration settings may for example not be the ideal after some time and thus need to be adapted according to management policies.

The driving force behind this concept is that adaptive management can thus lead to self-configuration, self-optimization and hence autonomy in network management. The key idea is to support self-configuration by being adaptive to varying conditions modeled as context, with high-level management policies driving self-configuration towards particular goals that lead to an optimal function of the network.

Nowadays networks and the number of services offered by them increase exponentially, therefore automation is required. In contrast to other management technologies such as management software or mobile agents, Policy-Based Network Management allows much faster modification of the requirements for the deployment and maintaining of a dynamic network. “For large networks with frequent changes in operational directives, Policy-Based Network Management offers an attractive solution, as it can dynamically translate and update high-level objectives into realizable network configurations” [11]. In this respect, a Policy-Based Network Management System has been developed in this thesis for the management purposes of the network overlay, because this particular way of management has the ability to adapt rapidly and efficiently to changing scenarios.

1.3. Goal

The research and the final outcome of the developed project are a proof of concept that Policy-Based Network Management is suitable for overlay networks and the evaluation of how such system would behave on a simulated network environment. This research problem can be formulated in the following questions:

- Are we able to build an autonomic self-managed system that is capable to react efficiently to environment (context) changes?
- How the performance of such system would behave over different network topologies and configurations settings?
- What are the policies that fit best to react on overlay networks with dynamic link changes, congestion, route changes, etc.?
- Can we completely take the network administrator out of the management loop?

1.4. **Methodology**

In this Master Thesis we developed a Policy-Based Network Management system to manage an overlay network in order to increase and improve the performance of the underlying network depending on the current monitored network conditions. The goal of this project is twofold. Firstly, an interface was designed to execute management and configuration tasks on the network overlay. The interface interacts with the SmartGRID simulator to support the modification of the values of the parameters of the algorithm during runtime. Furthermore, the real time behavior of the algorithm was observed in order to perform a sensitivity analysis. Secondly, a Policy-Based network management scheme was developed. The management policies that allow the dynamic and autonomic re-configuration of the network overlay when certain conditions are met were defined according to the sensitivity analysis. The outcome of this document includes a report with the state of the art on Policy-Based management of network overlays, the development of a Policy-Based network management system that interacts with a bio-inspired ant algorithm, and the evaluation of the developed system based on simulations in an overlay network.
As reflected in Figure 1, the research done in this thesis is two way, not only left but also right. The left part of the figure was done at the same time that the software was developed (right part). Each block represents one step closer to the final goal and outcome of the research: the development and evaluation of the Policy-Based network overlay management system. The research composes of the following:

- Literature review: study of the relevant work, reports and documents about Policy-Based approaches in network management, with special attention in the network management approaches in overlay networks and peer to peer networks.
- Study of behavior of the simulator: a sensitivity analysis is performed by running different simulations in the algorithm with modification of the parameters in order to evaluate the behavior of the algorithm.
- Definition of policies: different policies and possible scenarios are defined according to the results obtained by the sensitivity analysis.
- Evaluation of policies in the simulator: the stability and performance of the policies needs to be assessed in the simulator after the policy management system has been introduced.
• Identify the best policies depending on context information: tests were executed to evaluate which policies are better suited to apply depending on the context information, with the purpose of making the management solution autonomic.

The software development aspect of our current work composes of the following:

• Design of an interface to manage the network overlay: a graphical user interface was developed allowing the user to manage the simulator and control its parameters.
• Change of the parameters of the algorithm during runtime: the algorithm behind the simulator was modified supporting the modification of the values of its parameters while is running.
• Manual triggering of policies: the graphical user interface was extended introducing policies that change the behavior of the algorithm in the simulation when triggered manually.
• Introduction of context information on the simulator: a server was introduced to simulate the generation of context. When a change in the monitored context is detected, the algorithm modifies the values of the parameters throughout the simulation, trying to overcome context situations on a network such as load, delays and reduced link capacity.
• Automation of policies based on context information: the final solution is the Policy-Based network Overlay management system that reacts to changes of context information and automatically applies the defined policies that are best for reconfiguring the overlay depending on the current context.

1.5. Structure of the document

The structure of this thesis is as follows: Section 2 gives an introduction about the main concepts used throughout this document. Then it reviews the state of the art in overlay network management, focusing particularly on how to apply Policy-Based management to a bio-inspired overlay network developed by the PAI group, and how to apply, more concretely Policy-Based overlay network management. At the end of the section we analyze several papers with a related work on this research stream. In the Section 3, a sensitivity analysis of the algorithm is presented. The behavior of the algorithm was studied when different values for some parameters of the algorithm are modified. Then, different policies were defined to adapt to hypothetical scenarios. At the end we test the behavior of the algorithm when the parameters studied were modified during runtime. Section 4 details the different parts of software developed for the construction of the Policy-Based management system, their mission, functionality and features. Also, the methodology and technical aspects of the software development are discussed in this section. In Section 5 are defined the policies that govern the reconfiguration of the overlay network according to the context information. The results of the experiments performed are shown to prove the validity of the developed system, and the effects that the defined policies have in the overlay network are discussed. Finally, in Section 6 we can find the conclusions of this thesis, highlighting our contribution to the field of Policy-Based overlay network management with the solution that solves the initial problem, and the possible future steps in the direction of the current research.
2. State of the art

This section presents a state-of-the-art review summarizing the key concepts in the areas of overlay network and network management. The first part is mainly oriented towards overlay networks and the second towards network management in distributed environments. We particularly focus on bio-inspired overlay networks and policy based network management respectively, since the key concept and main contribution of this work is the autonomic policy based management of a bio inspired overlay network.

2.1. Overlay networks

2.1.1 Overview

Overlay networks are logical networks constructed on top of physical networks adding an additional layer for virtualization. They are used to increase routing robustness, security, reduce duplicate messages and provide new services for mobile users [12]. Among their applications we can find wide area routing architectures, efficient search of data items, selection of nearby peers, redundant data storage, hierarchical naming, trust and authentication, anonymity, massive scalability and fault tolerance [13].

Overlay networks can be classified in two categories: Structured and Unstructured [13]. In the structured ones, distributed hash tables (DHT) are used to construct the overlay. The structure of nodes in the overlay is done deterministically, each node maintains a set of links to other nodes, and for any key \( k \), each node either has a node identifier which owns \( k \), or it has a link to a node whose node identifier is closer to \( k \). DHT systems can guarantee that any node can be located in a small, \( O(\log N) \), number of overlay hops on average, where \( N \) is the number of peers in the system. These algorithms require probing between peers to determine latency. Some working implementations of structured overlay networks are Kademlia [14], Tapestry [15], ODRI [16], Chord [17] and P-Grid [18]. In P-Grid, for example, a tree data structure is used to store an associative array, where a node’s position in the tree shows with what key it is associated. All the descendants of a node have a common prefix.

Unstructured P2P systems are composed of peers joining the network with some loose rules; the resource discovery in this case is done by flooding techniques. The flooding techniques use time-to-live (TTL) to find the closer node in terms of internet metrics: each node probes the network to find its neighbors. Therefore, with flooding techniques there are chances that peers can become overloaded. According to [13] unstructured P2P overlay networks efficiently locate items but with higher overheads than structured ones. Some working implementations of unstructured P2P overlay networks are: Freenet [19], Gnutella [20], and BlatAnt [8], which is discussed in detail in section 2.1.2.

Overlay P2P networks are based on graph theory. Graph theory conceptualizes the network as a mathematical graph consisting of nodes (hosts) and edges (links between hosts). Graph theory helps in the construction or maintenance of the topology by looking for the optimal path through the overlay links from one overlay node to the other.
Due to the high heterogeneity in large networks, the routing path to communicate to the closest node of a group of computers in a network is not necessary the shortest. Overlay networks can be constructed in order to allow the sending of messages to destinations, which do not need to be specifically an internet address, in the shortest number of hops [21] [22] and furthermore, they can improve the performance in case of congestion detection, i.e. by routing through uncongested alternative routes using reactive routing [23] or even before congestion happens using proactive routing [24]. Overlay networks are also useful by providing management in large and complex network systems, because with the use of an overlay network, a complex network structure is abstracted into a simplified representation where multiple paths in the a network are grouped into a single path. This is especially useful in order to detect problems when managing large or complex networks [25].

Besides the useful applications already mentioned, the most important benefits of overlay networks are flexibility and scalability. Overlay networks are easy to deploy as they can be built without the involvement of network administrators or ISPs because these networks do not need modifications of existing hardware or protocols. Only the end hosts involved need to run the overlay protocol software to become part of the overlay network. Consequently it is just an incremental deployment of new technologies but without the need of a new planning or restructuring in the existing network. The cost of the mentioned benefits is an increased overhead because of the virtualization, because a new indirection layer is added to the network [12]. The complexity is therefore increased, as the new layer does not remove the complexity, but hides it.

Overlay networks topologies are evolving towards self-organizing systems. The research done at the PAI group in organizing overlay network topologies has led to the study of bio-inspired approaches. In the next subsection we introduce the basic concepts of the BlatAnt [8] algorithm: a bio-inspired method to construct overlay networks that imitates the behavior of swarm insects when traveling among nests. This algorithm belongs to a branch of artificial intelligence called Ant Colony Optimization, which already has proof of success in solving problems of distributed computing systems, load balancing and routing [26] [27] [28]. BlatAnt is described because we developed the Policy-Based management solution on top of a simulator of this algorithm, interacting with some elements of it.

2.1.2 BlatAnt

BlatAnt [8] is a fully distributed and adaptive algorithm for optimal construction and maintenance of overlay networks. BlatAnt is inspired by Ant Colony Optimization algorithms, and targets dynamic and evolving networks without requiring a global knowledge of the overlay structure. The key point and main feature of BlatAnt is that it creates and maintains overlay networks with small diameters. This goal is achieved by adding or removing logical links between the nodes within the overlay. The method used by the algorithm, rewires iteratively an existing network according to a set of user defined parameters to bounds the average path length of the routes between the nodes, recreating and demonstrating “the small world phenomenon” in an overlay network structure. The BlatAnt algorithm was created by Amos Brocco [48] in the PAI group and it was tested using a custom simulator which is the same environment in which this thesis has been realized.
The BlatAnt algorithm uses the bio-inspired concept of pheromones. Pheromones are chemical signals used by real ants to keep track of food sources or other ants. Among others, this bio-inspiration features the behavior of the path chosen by ant colonies when travelling among nests, finding food sources or transporting the food. BlatAnt simulates this behavior for discovering paths and creating overlay links between nodes in an overlay network. Ants in a computer network are lightweight pieces of software code that can perform actions on the nodes of the overlay network with the possibility of accessing and changing information on the nodes. The main mission of the ants is to collect localized information from the nodes and make it available to other nodes in the network in order to create optimized paths according to the specific parameters of the algorithm.

An important feature is that the algorithm is completely distributed. Each node executes the algorithm independently. There is no knowledge of the rest of the network from the view of each node; because of this, the job of the ants is to give each node the information about its neighbors in the network. The parameter ant vector length is a memory buffer that ants carry collecting information each time the ant visits a node, and dropping the oldest information when the buffer is full. The ant copies the information of the vector in the alpha table. The Alpha table stores information about the node’s neighbors like distance, identifier and the timestamp of the last update. The Beta Pheromone trail keeps track of the alive neighbors in order to detect the departure of a node from the network, and the Gamma pheromone trail increases the probability of a logical link to be visited by the ants in order to force a complete coverage of the network. Delta pheromone indicates the increment of the pheromones for each path on each travel of the ant.

The overlay network in BlatAnt is represented as a graph. The objective of reducing the diameter is achieved by rewiring the overlay network. The purpose of the rewiring is to minimize the diameter of the graph bounding its maximum value. This bounding process is mainly based on two rules: the connection rule and the disconnection rule. The connection rule adds a logical link between two nodes when the minimal distance is equal or greater than $2D - 1$, where $D$ is a user defined optimization parameter used to indicate the bound of the network. The disconnection rule removes a logical link whenever the distance is lower or equal than $D + 1$. When the rewiring process has reached stability, the diameter has an upper bound of $2D - 1$.

To accomplish the management of the overlay creation, the ant agents carry the activities required. Discovery ants wander randomly across the network collecting and spreading information about visited nodes, and updating the Alpha table of each node. Construction-Link ants are sent when a new node wants to connect to the overlay checking if the recipient has the maximum allowed links. Optimization-Link ants are sent aiming to optimize the routes, creating links between nodes to check if the connection rule is satisfied. Unlink ants are used to bound the network removing links to check the disconnection rule. Update Neighbors ants are generated when the neighborhood has changed, and they update the Alpha table in the new neighborhood. Finally Ping ants are sent periodically between nodes to keep the connection alive in case of low traffic situations.

The main difference among the working implementations for algorithms commented above and BlatAnt [8] is that in BlatAnt the global knowledge of the network is not needed.
Furthermore, there is no centralized information, making therefore the solution completely
distributed. Additionally the algorithm bounds the diameter of the network creating optimal
overlay nodes communication. These three characteristics make the BlatAnt algorithm an
attractive approach to build overlay networks.

2.2. Network management

2.2.1 Overview

While a network monitoring system monitors the activities in the nodes and network
connections to detect problems and avoid overloaded or crashed servers, overloaded
network connections or device problems, “a network management system refers to the
activities, methods, procedures, and tools that pertain to the operation, administration,
maintenance, and provisioning of networked systems” [29].

The term operation in the previous sentence means to assure that network and services are
working properly. It requires network and service monitoring to pinpoint a possible problem,
ideally before the network or service is affected. The administration part controls the
resources and how they are assigned. Maintenance involves repairs (when some network
element is damaged and needs to be replaced) and upgrades (if an existing device need to
be replaced with another one with higher specifications or if more elements needs to be
added to the network). Maintenance also keeps track of network configuration parameters
(such as delay, jitter or TTL) and optimizes them in an effort to make the managed network
run smoothly. Provisioning takes care of supporting new devices or services added to the
network.

FCAPS [30] is the ISO Telecommunications Management Network model and framework for
network management. FCAPS is an acronym for Fault, Configuration, Administration,
Performance and Security. These functional blocks are categorized in the ISO model defining
common network management tasks. Among others, important goals in network
management are to manage the resources and services, keep track of Quality of Service
(QoS), detect and fix errors and simplify the complexity of the administration of network
structures, especially in large and complex networks.

There are several software tools and protocols available to manage networks. Among
protocols, maybe the most used one by the manufacturers of network devices at the
moment is the Simple Network Management Protocol (SNMP) [31]. SNMP is based on UDP
and consists of a set of protocols for network management. In brief, SNMP consists of a
manager station, agents that are software installed to respond to queries from the manager,
MIB (management information database) which stores the collection of managed objects,
and objects which are variable that represent the status of an agent. Among software
solutions, there are commercial ones like HP Openview [32] which is widely used in large
corporations, and open source software alternatives like OpenNMS [33]. These two
mentioned software solutions are possibly the most popular alternatives to SNMP for
network management.
To accomplish the benefits that the overlay networks offer us, the overlay networks need to take into account management tasks. The management of overlay networks needs to deal with the following issues [12]:

- **Dynamic changes**: in distributed scenarios, dynamic changes in paths, changes in topology, routing, and links (especially considering mobile networks) may occur making the management information obsolete. These factors require a fast and optimal adaptation to maintain the QoS, throughput, and performance.
- **Resilience**: overlay network management should provide mechanisms for creating a robust network structure in the case of of nodes frequently joining or leaving the overlay network.
- **Global management**: as each overlay node has just a limited view of the rest of the network, overlay network management needs to coordinate communication among the nodes in the overlay.
- **Overhead**: the overhead is added due to the layering that the overlay network adds to the network communication. The overall performance of the network communication needs to be assured and, moreover, that this added overhead does not produce an overload in the underlying network.
- **Security**: in the case that confidential data is transported within the overlay, a security mechanism in the overlay network or between overlay networks should be implemented.

### 2.2.2 Policy-Based Network Management

The concept of policy management is usually used in security defining different rules for different types of users (administrator, user, etc.), to restrict the access to different resources. In network management, the different types of users can be viewed in this schema as the different state or conditions in the network. A list of desired parameters is defined into a rule set of actions to perform in the network devices. To make a clear example, in the case that a network administrator wants to optimize the performance in the network either because a service is degraded, or just to improve the current network performance, the network administrator does an enormous work configuring and coordinating the different network devices for performing actions that maintain the desired network parameters to tweak the performance on the network and achieve the desired goal. In Policy-Based network management, the events occurred, the conditions, and the actions to perform are defined into policies for an automatic management of the network simplifying a considerable amount of work for the network administrator.

A policy rule is modeled as an aggregation of policy conditions and policy actions. According to that representation, in its simplest form a policy rule expresses the following statement [34]:

*When (event) if (set of conditions) then execute (set of actions)*
The main advantages of Policy-Based approaches are: scalability, flexibility and simplicity [35]. Policies can help to automate the administration of overlay networks because of their flexibility. The separation of policies from the underlying managed system and the possibility to enforce a single policy on many devices and objects reflects the flexibility and scalability of using policies for overlay network management. Furthermore, policies can be enforced on a network entity to make changes in its configuration controlling the behavior in the network segment [36] making easier the work of the network administrator who can transform objectives into policies. Policy management systems therefore represent a suitable and efficient means of managing overlays. The use of policies in overlay networks allows flexibility and customizability for network management to control the network entities on the fly [37].

Because nowadays networks and services offered by them increase exponentially, the automation of network management is required. In contrast to other management technologies such as commercial network management software that uses SNMP or similar methods, requiring the probe of the network or mobile agents [38], Policy-Based Network Management allows much faster modification of the requirements for the deployment and maintenance of a network with dynamic changes. “For large networks with frequent changes in operational directives, Policy-Based Network Management offers an attractive solution, as it can dynamically translate and update high-level objectives into realizable network configurations” [11]. Policy-Based Network Management Systems are very popular nowadays for managing static networks; however, Policy-Based Network Management has been selected in this thesis for the management of dynamic overlay networks mainly because such schema can adapt rapidly to changing scenarios supporting flexibility and scalability.

In brief, a Policy-Based Network Management Systems (PBMS) can be described as the structure that reacts to the different notifications received, and when events occurred in the network and the conditions defined in the policy are met, the PBMS triggers the enforcement of the policy that leads to the desired network behavior.

According to the specifications of IETF [34], the schema of a Policy-Based Network Management is illustrated in Figure 2 and consists mainly of four components:
Figure 2: IETF Policy-Based Management architecture

- Policy Decision Points (PDPs): logical entities that are responsible for analyzing and handling events and making decisions based on those events.
- Policy Enforcement Points (PEPs): network devices that enforce policy decisions when a criterion is satisfied in response from the PDPs. The PEP collects the information about the network state and describes its network attributes or characteristics to the PDPs.
- Policy Repository (PR): memory where the policy rules are stored. In large systems the policies are usually stored in a database.
- Policy Management Tool (PMT): user interface for creating, viewing, modifying and enforcing policies.

A PDP manages one PEP at least, but usually it manages more. The PEP tells the PDP what actions it is capable of enforcing and the format of how it wants the actions represented in the policies. The PDP does the high level decision making based on the policies stored in the PR. The PDP is also responsible for translating the policies into a language that the network device can understand. PDPs and PEPs are logically separated, but that does not imply physical separation [39]. The communication between the PDP and the PEP is usually done by the Common Open Policy Service (COPS) protocol [59].

Because overlay networks can be distributed in Wide Area Networks (WAN), besides the described components of a Policy-Based network management system in an overlay network, the Policy-Based Overlay network management system needs to take into account the context information from different sources. Such sources include the service provider context, the network provider context and the user context as suggested in [36]. Changes in these different areas need to be studied in order to define policies when building a Policy-Based Overlay Network Management System (PBONMS) for adapting to the different network situations.

Adaptation is necessary during the lifetime of the overlay network to deal with new hosts joining the network, failures of nodes or congestion control and adaptation of routes [36],
especially in mobile scenarios. Furthermore, in some cases of high load, the sacrifice of terminating of some links could be necessary to ensure the overall performance, avoiding congestion and guaranteeing the QoS levels. We propose an example of an ideal case, with no delays and no transmission errors of an overlay network that is working at 100% performance, using the full capacity of the underlying links. In such example, if a node joins the overlay network, such process generates additional traffic in the underlying network. Such traffic could create queues, retransmission of packets and potentially loss of data. Policy-Based Network Management brings a fast reaction and a quick solving of such issues by applying policies to the network entities that could terminate some links when the load on these links reaches a certain threshold to avoid the mentioned problems.

There are commercial solutions that use Policy-Based network management approaches. Examples of such commercial solutions are HP PolicyXpert[60], Extreme Extremeware Enterprise Policy Manager [61] and Cisco Ciscoassure Policy Networking [62]. In [60] the Policy-Based network management is focused in controlling and prioritizing end to end QoS allowing the configuration of multiple heterogeneous devices and translating the policy information into the specific device configuration.

In this Master Thesis we developed a prototype of a Policy-Based Network management system for managing the overlay network that BlatAnt creates. The interface was developed following the architecture of Policy-Based Overlay Network Management system introduced in [34]. The developed system reacts to context information, and depending on the context information, adaptation is enforced in the overlay network that BlatAnt algorithm simulates; as a result, the overlay network changes its structure and behavior adapting its operation to high level requirements in order to improve the overall performance of the overlay network. The triggering of different events in the context memory automatically enforces policies which are processed by our Policy-based management solution. The management policies govern the different status of the network and the high level objectives defined by the network administrator. The interface designed is a distributed and autonomous system that does not require the intervention of a network administrator to perform management tasks.

2.3. Policy-Based Network Management related work

In this section we summarize the main ideas of the scientific papers studied for the development of this Master Thesis. We selected these papers as a representative work of Policy-Based schemes that are related with: network organization, context awareness, improvement of performance, automatic configuration and adaptive components. This decision was made according to the aspects that are more important in the development of our final software solution. Here, we made a review of the main ideas of different methods proposed for network management in overlay networks and distributed systems. The purpose of this section is also to give the reader an overview of what has been done in the fields related to Policy-Based Network Management at the time of writing.

An important aspect studied in this thesis is the effects produced in the network based on the construction of the network topology. The work by Di Ferdinando et al [40] discusses a first approach for the automation of an adaptive topology. The topology is adapted for an
optimal performance in peer to peer networks (P2P) based on policies and provision of resources. The solution is completely distributed and it is centered in the node. Each node acts independently and decides its neighbors based on its resources and a careful profile policy for delegation. All the management policies are broadcasted by means of flooding. Only the nodes that own enough reliable resources accept the policy and apply the actions.

Another aspect considered is the automation process of network management based on policies. In the work by Kowtha et Jiang [41] a finite state machine based framework is proposed for policy based management of complex networks, measuring the network behavior in each state and changing the policy according to each state and current network events and conditions observed in the network. The approach reduces the numbers of conditions in the policy rule sets, to efficiently predict and recover from events. In addition, because the number of states is finite, the network administrator could gain awareness prior to policy implementations. Figure 3 illustrates the original states as well, as the potential consequent and prior states.

![Figure 3: N-State machine for aggregated network conditions [41]](image)

Jones et al in [42] present a mechanism to manage overlay networks using Policy-Based methods. The solution allows multiple managers to play separate roles for the management of multiple overlay networks using a knowledge delivery mechanism. The mechanism is a middleware solution that publishes semantic information about the network (e.g. performance statistics) upon event occurrence, and stores a management information database. With an event-condition-action model the middleware triggers management actions when policies are met.

In the Celtic project Madeira [35] the policy based management of network elements in heterogeneous scenarios is addressed. This approach defines Adaptive Management Components (AMC), configuration management (CM) and Fault management (FM) entities distributed all over the overlay. These components in conjunction to policies provide a framework for network management applications by exploiting self-organization, symmetric communication and distributed control of P2P networks. The problem of heterogeneity is solved using Model Driven Architecture (MDA) [43] by means of web services, specifying and separating the logic and adapting data to be executed in different platforms and network elements.
An implementation of a management framework for mobile ad-hoc networks (MANETs) based on an automatic Policy-Based approach is presented in [44]. This work by Malatras et al. validates the enabling of self-management in MANETs. It supports self-configuration and adaptation by being aware of varying context conditions. A case study application is proposed to increase routing performance by real time adaptation of the evolving scheme. In this case every MANET node is equipped with sensors to monitor context. For example, to monitor the node surroundings, the location and speed is provided by a Global Positioning System (GPS) module. The routing strategy is based on policy rules driven by the relative velocity and location of the nodes and hence the mobility of the MANET.

Security is considered in the creation of some types of overlay networks. The work by Martínez et al [45] introduces a proposal to deploy secure overlay networks in multilayer networks. Security is assured by negotiating intra-domain, inter-domain policies, Security level agreements (SecLAs) and trust relationships. Because overlay networks in general are built on trusted domains and therefore the negotiation of parameters regarding security is public, the exchange of SecLAs is done using XML representation, and a common information model (CIM). In this architecture, the PDPs are in charge of granting or not the access for joining the overlay to a node. Figure 4 shows the three levels of security policy in the proposed overlay network. The three levels are the negotiation of parameters between client and security domain, two security domains and finally two clients.

A decentralized version of a Policy-Based Network Management System is described in the work by Ayari, Kamoun et al. [39]. In this document, QoS for Ad Hoc Networks is supported by extending the traditional PBNM system with Service Level Agreements (SLA). An interesting approach is discussed with the goal of adapting the centralized PBNM system into a distributed Ad Hoc approach. The goal is achieved by decentralizing the PDP and PEP using techniques of resource discovery in MANETs. A PDPs is controlling a domain that contains a group PEPs depending on a distance threshold from the PEPs to the PDP. In an interesting point, it is discussed how a PEP can change its PDP when moving to another
domain. This issue is resolved by extending the COPS protocol and adjusting dynamically the PEPs for each PDP. The PDP defines a domain that contains groups of PEPs, only selecting the PEPs depending on the closest distance to the PDP.

Another very important issue that this thesis discusses is the necessity of the network administrator in a Policy-Based network management system once the system has been deployed and the policies have been defined. The work by Samaan and Karmouch [46] discusses the dynamic creation of policies during runtime by learning from the current system behavior, and the adaptation of the policies to the changes of context and the dynamic requirements. The research also focuses on the flexibility, by giving the users or applications the possibility of specifying their requirements in order to define new high level objectives for each user level. Furthermore, the work describes the hierarchy of different levels of policies and the adaptation of them based on the hierarchical level and changes in the context. The proposed framework is evaluated by running experiments of automatic policy adaptation and analyzing the network performance obtained.

The changes in the network context are studied in this thesis. The Underlay Fused with Overlay can be an example of adaptation to network context and congestion avoidance in UFO [47]: using a resilient layered routing architecture provides explicit notification about network conditions. The system can improve the efficiency and scalability of routing in overlay networks by responding quickly to changing network conditions, avoiding the problem of link failures or excessive congestion, by anticipating to the emergence of such issues.

Finally, in the work by Al-Oqily and Karmouch [37] the architecture for a service specific context-aware Policy-Based managed overlay network is proposed. At the time of writing, this is the only specific work that discusses Policy-Based Network Management for overlay networks. In the schema proposed in [37], the previously defined Policy Decision Points (PDPs) and Policy Enforcement Point (PEPs) are defined as Overlay Policy Decision Points (OPDPs) and Overlay Policy Enforcement Points (OPEPs) respectively, and their mission is to cooperate and adapt dynamically the overlay network according to the preferences of the network administrator to control several parameters in the network such as bandwidth and delay. The COPS protocol is used to exchange policy objects between the OPEP and the OPDP in an even upper layer than the overlay network. OPDPs and OPEPs together with a Policy Generator (PG) and Policy Repository (PR) constitute the main components of the structure. The PG generates policies according to context information to accomplish overlay network objectives. The PG also adapts the policies on the PR learning from the behavior of the system. The OPDP receives the policies from the PG, then the PDP analyses the received policies and constructs the topology that meets the requested performance, checking that the desired goal doesn’t conflict with previous policies. The OPDP contacts the PR for retrieving and enforcing the policies as needed. The OPEP is the point where the policies are enforced. It is connected directly with the agents distributed through the network that are allowed to perform the actions for obtaining the desired results. When a change occurs in the network environment, the adaptation process is triggered.

In our work, we are considering the latter architecture for our implementation of the Policy-Based Network Management system, as to the best of our knowledge no similar work has
been published in this area. Figure 5 illustrates the architecture of an Overlay Policy-Based 
Network Management system, according to the work by Al-Oqily and Karmouch [36], 
showing the main components of the proposed architecture and the connections among 
each component.

![Overlay Policy-Based Network Management architecture](image)

Figure 5: Overlay Policy-Based Network Management architecture [36] according Al-Oqily and Karmouch

Our research group, the Pervasive and Artificial Intelligence group of the University of 
Fribourg, has already done research in the area of pervasive computing developing a mobile 
agent routing platform for multi-service networks [38]. The BioMPE project is the focus of 
the current research: a unified pervasive computing middleware to address the issues of 
P2P overlay networks in the context of heterogeneous underlying network infrastructures. 
To increase the robustness, advanced node and network monitoring mechanisms are 
introduced, based on biology-inspired algorithms. Police-based network management in 
overlay networks is considered for several scenarios, one of which is the SmartGRID [9] 
project.
3. **Sensitivity analysis of BlatAnt**

In this section, the results obtained by a sensitivity analysis of the BlatAnt algorithm are explained. This analysis was done in order to study the behavior of the algorithm and find the policies that best manage the different situations that can occur on an overlay network such as the one that BlatAnt constructs.

When conducting the experiments, we run 10 simulations of 600 iterations of the algorithm for each parameter, to make statistically correct conclusions. We analyzed 4 parameters: \( D \), *Ant Vector Length*, *Alpha Table Maximum Size* and *Ant Birth Probability*. 3 different values were used for each parameter. Moreover, 4 topologies were analyzed: a grid of 32 x 32 nodes, an unstructured LAN network topology of 1281 nodes, a bus topology of 1024 nodes, and a topology in form of hyper cube with 1024 nodes.

It is noteworthy to mention that just analyzing the LAN topology for the configuration stated above lasted 15 hours and 50 minutes on a single computer (Core 2 Duo at 3 GHz and 4 Gigabytes of RAM memory). It is for this reason that we opted for using the cluster of the university (8 blades at 2.83 GHz, and 16 Gigabytes of RAM memory) to run our experiments.

### 3.1. **Comparison of performance in different topologies**

The simulations shown below were done using the default values for the parameters chosen \( (D=6, \text{alphatable}=20, \text{LV}=15, \text{birthprob}=0.15) \). As observed in Figure 6 and Figure 7, at the same time that the diameter is being reduced, the number of edges grows fast until the diameter reaches the limit of \( 2D - 1 \). At this point the algorithm converges, and the number of new edges stops increasing exponentially. The topology then reaches stability.

![Figure 6: Comparison of diameter differences in the different topologies with the algorithm in the default values](image-url)
As seen on Figure 6, the reduction of diameter on the first iterations is influenced by the initial topologies. It is logical to think that under the same circumstances, a graph with less diameter in the initial topology (i.e. in the hypercube topology the initial diameter is 10) will converge faster compared to the bus topology that has 1023 as initial diameter. A side effect can be seen on Figure 7 where we can see that the increase of edges on a hypercube topology is not as proportional as in other topologies, because in the initial graph on a hypercube the nodes on the network are regularly connected to each other.

![Figure 7: Comparison of the number of edges in the different topologies with the algorithm in the default values](image)

**3.2. Comparison of parameters**

For the creation of policies, we mainly based our sensitivity analysis of the BlatAnt algorithm on the results obtained by the simulations performed on the LAN topology, as this topology is the one that best reflects the structure of the underlying topology on top of which the overlay networks are usually built.

**3.2.1 Alpha Table Maximum Size**

An increase in the *Alpha Table Maximum Size* parameter increases the consumption of computer resources in the simulator, and on a real network “the alpha table should be limited to only the storage and computational capacity of each node” [8] because each node has to compute the distances among its neighbor nodes. If the value of the *Alpha Table* is increased, every node can have a wider view of the network as it can store information about more nodes in the overlay. The *connection rule* of the algorithm is therefore performed with more information, so more paths that lead to optimization will be created and hence fewer removals of links (*disconnection rule*) are performed. Because of this, the
increase appreciated in the graph for the edges is not as exponential as with other parameters.

*Alpha table Maximum Size* is the parameter that, when increased creates fewer edges while reducing rapidly the diameter of the network graph, in comparison with the rest of the parameters analyzed.

![Figure 8: Evolution of diameter for different values of Alpha Table](image)

Master thesis: Policy-Based Management of Overlay Networks
In Figure 8 it is observed that for a lower size of alpha table, the algorithm converges slower (more iterations are needed to reach the expected diameter and stability on edges creation). The reason for this behavior is that as less information is stored in the alpha table, less combinations of pairs of nodes are stored, and therefore potential routes that lead to a decrease of the diameter could be removed from the alpha table. The routes that lead to optimization are less if each node has less information about other nodes, so reaching stability with less capacity of the alpha table in the nodes is a longer process.

Figure 9 shows that a bigger size of alpha table reduces the creation of edges, while still reducing the diameter. Additionally, a higher value of the parameter for the alpha table size performs especially well on the first iterations of the algorithm when creating the overlay network. In the opposite case, because of the reasons explained above, a lower alpha table size will show a faster increase of edges with a lower reduction of the diameter. After all this reasoning we can state that in order to rapidly build optimal overlay links in the overlay network that BlatAnt algorithm creates, a bigger size of alpha table is preferred.

3.2.2 Ant Vector Length

The ant vector length (LV) is the parameter used to indicate the length of the vector that the ant carries. The ants are collecting information about the visited nodes in this vector, and every time an ant visits a node, the ant updates its alpha table with the information it carries about the visited nodes. When the vector is full, the ant drops the oldest information. Therefore, the larger the size that the ant vector length buffer has, the more information
about the visited nodes is carried and the alpha table (when big enough to store all information of the vector’s) has more information about the rest of the overlay nodes and therefore, the possibility to construct new overlay links is also increased as observed in Figure 11.

$LV$ ideally should be higher than $2D - 1$; if not, the ants cannot collect enough information to construct an overlay network given $D$ as optimization parameter. This is due to the structure of the algorithm: if a node has information about the $N$ last visited nodes, the view of each node in the path which the ant has updated the *alpha table* is limited to N-1. If the ant has a lower buffer, it cannot store all needed information from the visited nodes and therefore, the information regarding the *alpha table* of the overlay graph and connections cannot be enough to build a network with a diameter less than $2D - 1$. Therefore, the *ant vector* length needs to be big enough to construct the overlay network with the desired optimization parameter.

As observed in Figure 10, an increase of the *ant vector length* decreases the number of iterations needed to reach the convergence of the algorithm. Note that an increase of the $LV$ parameter also implies that more information is transported on the network, so a compromise between the size of $LV$ and the used bandwidth is needed.

The $LV$ parameter is the one from all analyzed ones that performs the convergence of the algorithm faster (if higher values of $LV$ are set). But it is also the parameter which creates more edges compared to the rest of the parameters analyzed in this sensitivity analysis.

![Figure 10: Evolution of diameter for different values of Ant Vector Length](image-url)
3.2.3 Optimization parameter D

Higher D parameter allows for a bigger diameter in the network, so for the nodes which already have this diameter with the rest of the neighbors, theoretically no new edges need to be generated. Because of this observation, we can say in general that if we are more permissive with the diameter on the overlay network, fewer edges need to be generated. Therefore, for a bigger value of D fewer edges are created.
We observe in Figure 12 that for the D parameter, values less than 6 very slightly affect the behavior of the algorithm (number of edges and iterations needed) compared to the results obtained with the value 6, which is the default parameter. After observing that there are almost no differences with lower values of D, it makes us think that the algorithm cannot perform better for a lower diameter in the first iterations. However, when the algorithm has reached the stability, the diameter should be reduced if the D value is lower than 6 with the assumption that there is also, a big enough alpha table size, and vector length. Figure 13 shows that for high values of D the increasing number of edges is less exponential and more linear. We also observe that once the overlay network has reached the desired parameter the number of edges stabilizes.
3.2.4 Birth Probability

For a higher value of the birth probability parameter, more discovery ants are created, therefore more information is collected, and this effect will highly likely lead to new edges being generated. Therefore, the probability that these discovery ants will find useful routes is also increased, this is especially useful on the first stages of the algorithm, or when after a high number of new nodes joins the network. This phenomenon can be observed more clearly in a bus topology, where the connectivity among nodes is very limited and the algorithm performs especially well by dramatically reducing the initial diameter after only 100 iterations (see annex 3.3 for details).

We observe also the same effects of the birth probability parameter but on a lower scale for the LAN topology, which is the one analyzed in this section. Figure 14 and Figure 15 show the results of the simulations with different values of birth probability, and the effects on the diameter and edges can be noticed. We first mentioned the example of the bus topology because the effects of increasing or decreasing the birth probability parameter are better seen in this topology.

One of the consequences of increasing the birth probability is that more ants are generated. An increase of birth of the ants generates more control traffic in the network that could be prejudicial whenever the network is in high load or needs high QoS.
Figure 14: Evolution of diameter for different values of Birth Probability parameter

Figure 15: Evolution of edges for different values of Birth Probability parameter
3.3. **Summary of the sensitivity analysis**

The experiments done have shown that there is a noticeable change on the behavior of the topology maintenance of the overlay network depending on the values of the parameters analyzed. The motivation behind the sensitivity analysis of the BlatAnt algorithm was to identify the influence of the variability of the parameters of the algorithm in order to know which parameters improve the performance of the algorithm in certain situations. With this knowledge we are able to create policies that modify the behavior of the algorithm to create or optimally maintain an overlay network.

In particular, we have proven that to speed up the convergence of the algorithm on similar network topologies, an increment of some parameters, like \( LV \) is beneficial if the network has available resources, and if the \( \text{alpha table} \) is large enough.

One of the effects of the algorithm is an excessive creation of edges while bounding the diameter; this excess of new edges can be translated as alternative routes which serve as a backup whenever a node leaves the network, or some link is down.

The results of the sensitivity analysis can be summarized in the following:

- \( \uparrow\uparrow \text{alpha table} = \downarrow\downarrow \text{diameter fast. (Needs capacity on the network nodes).} \)
- \( \downarrow\downarrow \text{alpha table} = \text{reduce the probability of the node to have optimal paths to the rest of the nodes in the network.} \)
- \( \uparrow\uparrow \text{LV = \downarrow\downarrow \text{diameter fast. (Implies more traffic on the network).}} \)
- \( \downarrow\downarrow \text{LV = reduces the view of each node in the overlay network.} \)
- \( \uparrow\uparrow \text{D = \downarrow\downarrow \text{number of edges.}} \)
- \( \downarrow\downarrow \text{D = stabilizes the number of edges.} \)
- \( \uparrow\uparrow \text{birth probability = stabilization of the network = \uparrow\uparrow \text{control traffic.}} \)
- \( \downarrow\downarrow \text{birth probability = worse performance of the algorithm in unstable networks due to lack of discovery of working paths.} \)

3.4. **Scenarios and policies**

In this section we introduce the policies that can be applied on the overlay network to reach certain goals in the underlying network in different situations, based on the aforementioned analysis. However, the different scenarios would need to be additionally studied on the specific underlying network for concluding the best policy rules to define depending on the different network conditions.

In the case that we want to reduce the excessive creation of new links on the overlay network, (a stabilization of the increasing number of edges on the network graph is desired) because the total number of connections of the nodes in the network is already very high and the computation in the nodes and the complexity of the management is therefore increased, an increment of the value for the \( D \) parameter can be enforced on the simulation of the algorithm. On the contrary, if a reduction of the diameter is desired, the value of \( D \) needs to be adjusted back to a lower level.
If we want to achieve a scenario with high robustness (for example a hypercube topology, where we have many alternatives for the same route if a link on the network breaks) an increase of *ant vector length* will help to bring stability to the network faster. This effect is guaranteed because the number of edges is increased and backup routes are available. This policy can be useful in case of a critical node crashing or in a situation of instability (nodes leaving and joining the network, temporary unavailable links, etc.). The effect of the exponential growth of overlay link studied in the sensitivity analysis would lead to a fast restructuring of the overlay.

In a situation of stability (fixed and always-on nodes and the underlying network structure is static) if each node could have the maximum information possible about the rest of the nodes in the network, the overlay creation would be optimal. Therefore an increase of the *alpha table maximum size* gives more information about the network to each node, reducing construction or maintenance of the overlay links and creating only the necessary paths to achieve the desired diameter in the overlay network.

When a part of the network has been disconnected, the overlay links of the network would need to be restructured. A temporal increase of the *birth probability* parameter would be beneficial in order to make new overlay links to overcome the non-used ones, making the structure stable and maintaining a small diameter after a network split.

Another policy we could use to avoid the overload of the network is to decrease the rate of creation of new ants. A low value of *ant birth probability* reduces the control traffic on the underlying network. In a case of best effort, the network may be working at full capacity, and then control traffic in this situation could be prejudicial for the overall performance of the network since control traffic adds data packets to the network.

Combination of more than one parameter to achieve specific goals could be done in order to create new policies. However this would require more experimentations and study of the behavior of the algorithm when the combinations of different parameters and different values are applied.

### 3.5. Modification of parameters during runtime

In this section we show the results obtained performing modifications of some parameters of the algorithm during runtime. The idea behind these experiments is to show the reaction of the overlay network when applying such modifications, which could be abstracted into simple policies.

For the execution of these experiments we had to modify the core of the algorithm for allowing the modification of the parameters during runtime. In the original version of the algorithm it is not possible to change the values of the parameters of the algorithm once the simulation has started.
What we will see in the following charts is the reaction of the network when the actions of certain policies are manually applied. The charts show a comparison of the edges and the diameter changes for the LAN topology for different values of each parameter compared to the default values. After the 200\textsuperscript{th} iteration the default values are modified. We compared the behavior of the algorithm after the modification of the parameters with the results obtained by the sensitivity analysis for the same configuration settings. We conclude that all graphs match the pattern of the results obtained by the previous sensitivity analysis.

3.5.1 Alpha table maximum size

![Diagram showing the comparison of diameters for different Alpha Table Sizes](image)

Figure 16: Diameter comparison between a fixed value of Alpha Table Size and the same value but changed after the 200\textsuperscript{th} iteration
3.5.2 Ant vector length

Figure 17: Edges comparison between a fixed value of Alpha Table Size and the same value but changed after the 200th iteration

Figure 18: Diameter comparison between a fixed value of Ant Vector Length and the same value but changed after the 200th iteration
Figure 19: Edges comparison between a fixed value of Ant Vector Length and the same value but changed after the 200th iteration

3.5.3 D parameter

Figure 20: Diameter comparison between a fixed value of D parameter and the same value but changed after the 200th iteration
3.5.4 Birth Probability

Figure 21: Edges comparison between a fixed value of D parameter and the same value but changed after the 200th iteration

Figure 22: Diameter comparison between a fixed value of Birth Probability and the same value but changed after the 200th iteration
It is noteworthy to mention that while an increase of birth probability parameter during runtime does increase the number of edges, and a low initial value of the birth probability does create less edges than higher values, several experiments done have shown us that a decrease of the value at some advanced iteration in the simulation does not affect substantially the reduction of edges. Therefore we have decided not to create a policy based on decreasing of the birth probability.

3.5.5 Discussion

With the run of this series of experiments we have the proof of how the network overlay dynamically changes its behavior by modifying the parameters studied in the sensitivity analysis. The results of these simulations guided us in defining the policies to manage the overlay network that BlatAnt creates according to our preferences or the network needs, and to develop the software for Policy-Based Management, as presented in a later section.
4. Design of a Policy-Based overlay network management system

We present the design and development of a software solution following the guidelines and requirements of the theoretical aspects of the Policy-Based network management schemes studied. Our works serves as a proof of concept that a Policy-Based network management system can work on an overlay network. In this section we details on: the different pieces of software developed to build a Policy-Based overlay network management system, the choices in the software designed and the methodology used to accomplish the goal. Section 4.1 states the requirements of the developed system, while section 4.2 describes the designed system and the different parts of software needed to achieve the goal of this project. In section 4.3 are described the software tools used for the development of the different elements of our system. Throughout Section 4.4 we show and explain the functionality and features added to the different software pieces.

4.1. Requirements

4.1.1 Scope

The practical part of this Master thesis has an objective: the design and development of a Policy-Based overlay network management system. The goal of the developed management system is to optimally reconfigure the overlay network according to changes in network context.

A Graphical User Interface (GUI) was created to easily perform changes in the configuration of the BlatAnt algorithm simulator and to obtain information and results of the different simulations. With the help of this GUI, a sensitivity analysis of the algorithm presented in the previous section allowed us to observe the changes in the creation and maintenance of the BlatAnt overlay network. In this first approach, we familiarized with the software simulator and furthermore we hide the complexity of a text user interface simplifying the interaction with the algorithm, helping the user to understand the inputs and interpret the results.

In the final stage, a Policy-Based overlay network management system was added to the GUI. The software solution is able to execute management tasks on the overlay network simulator, allowing the dynamic and autonomic reconfiguration of the network overlay to achieve high level objectives, based on the defined policies. Additionally, the administrator can define management policies in a Policy Management Tool, and such policies are triggered when changes of context are detected and certain conditions in the running simulation are met.

4.1.2 Conceptual structure

The conceptual structure of the Policy-Based overlay network management system traditional developed system is shown in Figure 25. At the left side of the figure we can observe the PBNM architecture. In the right side the conceptual structure for the specific developed solution is presented; the different boxes represent the software which runs on
each machine and how the different machines are connected which runs each piece of software.

![Logical PBNM Architecture](image)

**Figure 24: Conceptual Structure of the developed system**

### 4.1.3 Typical scenario

We have defined and explained in Section 2.2.2 the different elements that compose a Policy-Based network management system: PDPs, PEPs, PR and PMT.

However, to have a better understanding of the high level goals of each policy, we have illustrated a possible scenario in Figure 25, where the different elements and the location of each element are seen. In the figure, we can see an ISP and a regional network of a town (for example, Fribourg), connected over the Internet. On the Fribourg network, different PEPs can be located across the network measuring relevant context information about the Fribourg network; the PDPs queries the PR located on the ISP side on every change of context to retrieve the list of policies that match the current situation. When the PEPs send the current network device parameter conditions to the PDP, and the conditions match the defined condition in the PR, the PEP enforces the actions defined in the policy. The actions to enforce vary according to the high level objectives defined by the network administrator in the policies. The name of each policy is representative of the high level objectives that are pursued with the enforcement of the specific policy.
4.2. System design and software development

We detail here the key points and most interesting features of the software developed for our Policy-Based Network Management System.

4.2.1 System architecture

We designed the different software entities to act independently so that they can be located in different parts of the network making the solution distributed. Figure 26 illustrates the different pieces of software and the type of connections between them. The text inside the parentheses defines the function of each software entity related to the Policy-Based
Management scheme. The different software parts, with the exception of the specific PEP of BlatAnt could be relatively easy adapted to work on different overlay networks.

![Logical structure of the different software programs](image)

The different pieces of software and their functions are explained below:

- **Context Generator**: simulates the change of context. It sends data over a socket connection to the PDP with different content depending on the context simulated.
- **PDP**: monitors the context information and queries the policy repository on every change of context to know whether there is a policy to implement on the current context. When the query is positive, it requests the status of the algorithm execution from the PEP, and if the answer satisfies the condition of the policy, it sends the order to the PEP (PolAnt) for enforcing the actions stated in the policy.
- **Policy Editor**: is the Policy Management Tool. Retrieves, pushes and edits the policies into the policy repository database.
- **Policy Database**: Policy Repository, where the policies are stored.
- **PolAnt**: acts as PEP. Listens to socket connections from the PDP; when the PDP asks for the value of a running parameter, it check first in the active BlatAnt simulation the parameter asked by the PDP and then sends its value over a socket. When the query has been replied, the PDP can ask to enforce the actions stated in the policy, and then, PolAnt enforces the retrieved actions modifying the parameters of the current simulation.
Because the Policy-Based Overlay Network Management scheme we developed is managing the simulated overlay network that BlatAnt creates, and at the time of writing, there is not yet access to the real network, we have chosen to simulate also the context information with a software program. In a real overlay network the PDPs make decision on the configuration of the overlay based on the context information that the PEPs are reporting to the PDPs. PEPs change the configuration on selected network devices when it is requested by the PDPs. The Policy-Based Management software scheme developed is fully functional and with a few adaptations, this solution could be used for a Policy-Based Overlay Network Management System that would work on a real overlay network.

4.2.2 Flowchart

The typical flow of events in the system is as follows: all software is launched (the Policy Editor not essential, but the database needs to be active, and the policies defined) and the simulation of the BlatAnt algorithm is started. The user can interact with the Context Generator to simulate a change of the context. When the user simulates a change of context, the PDP senses this change of the context and checks whether there is a policy in the PR that is related to this change of context. If the result is negative, the PDP just keeps listening for new events, when the result is positive it queries the PEP to retrieve the actual parameters on the network, and the PEP replies with the requested parameters. If the parameter satisfies the stated condition in the policy, the PDP sends an order to the PEP to enforce the actions defined in the policy. At any moment, the user can browse and modify the policies in the Policy Editor, to make changes in the existing policies by modifying the conditions or the actions to enforce in the overlay network. Figure 27 shows the described events in a flowchart.
4.3. Development tools and techniques

In this subsection we detail the different components and libraries needed to achieve specific goals in the software implementation of our Policy-Based Network Management System.

For the development of the main software components, the Java programming language was chosen. The reasons for this selection were mainly because Java is a standardized and can be easily used for development, with a big community and a rich set of libraries. It adapts to the hardware resources and it is multiplatform, which means that it can run on top of multiple operating systems. At the time of writing the latest version of the Java programming language was the version 6, so it is recommended to use this or a later version to execute the solution.

As a development framework, we have used the NetBeans [53] platform because it is a popular open source complete solution to write, compile and run Java programs. Its visual editor was very helpful for the development of the GUI (Graphical User Interface).

For the development of the graphical interface we have used a MVC (Model-View-Controller) design. MVC is a model of software architecture that separates the application data, the user interface and control logic into three distinct components. MVC contributes to the program flexibility and extensibility. For the development of the GUI, we have chosen the Swing [54] package of Java, because it encompasses many good features adding rich
graphics functionality and interactivity to Java applications. These features include GUI components, look and feel support and a Java 2D API which we used for the extension of the GUI.

The different pieces of software are packed in JAR file which contains all the classes, libraries and metadata necessary to run the programs.

4.3.1 Multi-threading

Because the BlatAnt software was initially intended to take the parameters relative to the simulation only from the command line, the structure of the main class, and some subclasses needed to be modified in order that the GUI is able to interact with the algorithm and display real time information.

Furthermore with a single-threaded GUI application, the intensive processing performed in the computer when BlatAnt algorithm is running freezes the GUI when the algorithm is running (user is unable of changing the value of a check box or go to a different tab), and the output of the algorithm cannot be displayed until the algorithm would be completely finished. This makes it impossible to view the real time status of the parameters on each iteration of the algorithm until the simulation has been finished. The GUI application was designed using different threads. We followed the Java recommendations for programming GUI with lengthy computations:

“When writing a multi-threaded application using Swing, there are two constraints to keep in mind:

- Time-consuming tasks should not be run on the Event Dispatch Thread. Otherwise the application becomes unresponsive.
- Swing components should be accessed on the Event Dispatch Thread only.” [55]

4.3.2 Sockets

To send information between the different software elements we have chosen to use Java sockets [56]. A socket is a two way communication link between two programs, a client and a server running on a network. The communication is done by Transmission Control Protocol (TCP) which provides a reliable, point-to-point communication channel for client-server applications on the Internet.

4.3.3 Database

For storing the information in the policy repository, we have chosen a MySQL [57] database. MySQL is a relational database management system that runs as a server providing multi-user access to a number of databases. The reason for using MySQL was mainly because it is free and open source and meets the requirements of the functionality needed for the Policy Repository.

The database has only one table “rules” where all policies are stored. The structure of the table is described below. It is needed to mention that the field “name” is the unique
identifier of each row; this means that 2 policies cannot be stored with the same name. The fields “ifx” and “whenx” are indicating the condition for the policy to be enforced, and the occurred event, respectively.

**Table structure for table “rules”**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>dparameter</td>
<td>int(11)</td>
</tr>
<tr>
<td>alphasize</td>
<td>int(11)</td>
</tr>
<tr>
<td>vectorlength</td>
<td>int(11)</td>
</tr>
<tr>
<td>birthprobability</td>
<td>Double</td>
</tr>
<tr>
<td>ifx</td>
<td>varchar(20)</td>
</tr>
<tr>
<td>whenx</td>
<td>varchar(20)</td>
</tr>
</tbody>
</table>

The content of the table rules, where the initially defined policies are stored is described below. Each policy changes the parameter only if it satisfies the condition for a policy i.e. the policy “Regeneration” changes the birth probability parameter to 0.2 only if the current name for this parameter is less or equal than 0.15. Despite the fact that all policies are changing just one parameter, the possibility of changing 2 parameters is also supported.

**Contents of table “rules”**

<table>
<thead>
<tr>
<th>Policy Name</th>
<th>Dparameter</th>
<th>Alphatable</th>
<th>LV</th>
<th>Birthprob</th>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regeneration</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.2</td>
<td>Birthprob.&lt;=0.15</td>
<td>Network Split</td>
</tr>
<tr>
<td>Reduce Connections</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Dparameter&lt;=6</td>
<td>High nr. of Edges</td>
</tr>
<tr>
<td>High robustness</td>
<td>0</td>
<td>0</td>
<td>19</td>
<td>0</td>
<td>VectorLen.&lt;=15</td>
<td>Attacks</td>
</tr>
<tr>
<td>Stability</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>Alphatable&lt;=20</td>
<td>Links are stable</td>
</tr>
</tbody>
</table>

### 4.3.4 Database connector

So that the different Java programs can communicate with the MySQL database, the Java Database Connector (JDBC) [58] was chosen. The JDBC API provides universal data access from the Java programming language to easily integrate with our database. The sending and retrieval of information and the operations performed in the database from a Java program is easily handled by JDBC.

### 4.3.5 BlatAnt

To simulate the overlay network and see the behavior of the overlay network when the different policies were applied we used the BlatAnt algorithm simulator. The BlatAnt algorithm simulator was created by Amos Brocco in the context of the SmartGrid project [48]. The implementation of the BlatAnt algorithm is presented as a Java [49] program able to simulate the creation and maintenance of an overlay network for any topology in an underlying network. The software requires at least 2 parameters to run: input graph and output file. As an input, the algorithm accepts a specific format of network graph topology file. As output, the program prompts the location of the file and directory where the
computed result files will be saved. The output file keeps the same format as the input graph file. Therefore, it is possible to load the output graph files as input files. The generated result files display the overlay network topology, optimized according to the BlatAnt rules. Table 1 shows an example of a network graph file that BlatAnt generates.

```python
!!python/object:networkx.graph.Graph
adj:
  0:
    1: null
  1:
    0: null
  2:
    1: null
    3: null
  3:
    2: null
    4: null
  4:
    3: null
  5: null
```

Table 1: Sample of input or output network graph file

The input/output network graph file we can see above is formatted in YAML [50] for the NetworkX [51] Python [52] package, and represent each node with a number and also, the connections from each node to its neighbors. We can observe in the graph that the nodes are only aware of their direct neighbor’s nodes, being unaware of the existence of the rest of the nodes in the network.

The software is iterative, namely when it is running, it displays the variations of some of the parameters. And during execution, it also saves iteratively the result graphs of the different topologies formed until the simulation is finished. Table 2 shows a sample output of the BlatAnt algorithm simulation when it is executed from the command line.

```bash
~/Blatant$ java algo.Algorithm --initial path-50 --prefix simul01
---------------------------------------------
0 A=9 Bt=8013 Rsp=0 E=98 +=0 -=0 dr=0 F=0 FF=0 B=0 LV=21.06
20 A=13 Bt=0 Rsp=2 E=98 +=0 -=0 dr=0 F=0 FF=0 B=0 LV=21.06
40 A=13 Bt=0 Rsp=30 E=142 +=25 -=3 dr=0 F=0 FF=0 B=0 LV=21.06
60 A=14 Bt=0 Rsp=6 E=136 +=1 -=4 dr=0 F=0 FF=0 B=0 LV=21.06
80 A=14 Bt=0 Rsp=5 E=136 +=3 -=3 dr=0 F=0 FF=0 B=0 LV=21.06
100 A=15 Bt=0 Rsp=5 E=136 +=1 -=1 dr=0 F=0 FF=0 B=0 LV=21.06
120 A=14 Bt=0 Rsp=2 E=128 +=0 -=4 dr=0 F=0 FF=0 B=0 LV=21.06
140 A=14 Bt=0 Rsp=2 E=122 +=0 -=3 dr=0 F=0 FF=0 B=0 LV=21.06
160 A=13 Bt=0 Rsp=1 E=120 +=0 -=1 dr=0 F=0 FF=0 B=0 LV=21.06
(…)
```

Table 2: Sample output of the BlatAnt software implementation

In the output, we can see on each iteration some parameters, like the number of ants (A), ant birth rate (Bt), number of edges (E) and the average length of the vector (LV) among others.
4.4. **Software elements**

In this subsection we detail on the different pieces of software developed and their main features. This section can also be used as a guide for the user of the software.

4.4.1 **Context**

In Figure 28 a screenshot of the software for the simulation of the context is shown. The usage is quite simple: when the user clicks in one of the four buttons, the context described in the text at the right of the button is simulated in the overlay network.

![Figure 28: Software for context simulation](image)

The software also gives the option for configuring the host and the port to send the socket connections. Figure 29 and Figure 31 shows how to access the configuration menu and the menu itself.

![Figure 29: Configure menu](image) ![Figure 30: Configuration window](image)

Internally, this program basically opens a socket connection and sends a different content to the host defined on the configuration window.

4.4.2 **PDP**

The software developed for the PDP is shown in the Figure 31. All components are visible on the main window. The software has a field to configure the port in which the socket is listening to the data that could indicate a potential change of context, a second field to configure the location of the database URI (Policy Repository) and two more fields to configure the host and port, for opening a socket connection to the PEP. When the start button is pressed, the PDP is ready to listen to changes from the context simulator and send connections to the PEP.
4.4.3 Policy Editor

The Policy Editor is the software built as a Policy Management Tool (PMT). It is the user interface that the network administrator uses for defining, modifying and removing the different policies. This software is basically a GUI for the user interaction with the database. A screenshot of the software is shown in Figure 32. In the upper part of the main screen we have a copy the contents of the database displayed on a table; and in the lower part we have intuitive fields, combo boxes and buttons to perform operations in the policy list.

When the button “New” is pressed, a new row is added to the table and all components (spinners, combo boxes and text fields) are cleared. As soon as the fields, combo boxes or spinners are modified, the contents of them are updated in the table row, but until the button “Save” has been pressed, the contents of the row are not written into the database. Figure 33 shows a screenshot of the reaction of the software just after the new button is pressed. To save a new policy we must write the name, and select values from the “if” and “when” combo boxes as minimum requirements. If the value of some parameter to enforce (Dparameter, Alphasize, Vectorlength and Birthprobability) when defining the policy is set to 0, the value of that parameter in the current simulation will not be modified.
Because the context information is simulated, we limited the field “when” to the four different situations defined on the context. In the field “if” the user has more freedom as he can define its own rules; the two combo boxes, the spinner and the button let the user define new conditions for different parameters with different values. When the button marked with the button “add” is pressed, a new condition rule is added at the bottom of the list in the field “if” and it is automatically selected for the definition of the actual policy. The software can also recognize the correct data type when selecting the value of the parameter in the “if” condition and it is always parsing the integer part of the value if the parameter just admit integer numbers (as in the \( D \) parameter) and a double value if the parameter only admits real numbers (as in the birth probability parameter). Figure 34 shows the screenshot of the user interface in which the user can define new conditions for the “if” part of the policy.

The button “refresh” on the Policy Editor refreshes the content of the table with the contents of the database. This can be especially useful in the case of a database or application crash. As soon as the database is ready, the contents of the database can be refreshed using this button. Also, if new policies are added manually in the database, manually or with another instance of PMT, the content can be updated. The button “delete” removes rows from the table, but the policy is not deleted from the database until the button “save” is pressed.

It is noteworthy to mention that in the case that 2 policies are matching the conditions “when” and “if”, the policies are sent to the PEP at the same time, but if both of them are modifying the same parameter, the policy which is higher in the list of policies is applied first in the current simulation. That means that if both policies are modifying the same parameter, the PEP will enforce the values of both policies but the values of the parameters defined on the second policy are the ones which will be kept for the remaining of the simulation.
4.4.4 PolAnt

PolAnt is presented as a GUI for configuring the parameters and running the BlatAnt algorithm simulator, allowing different ways of displaying the results. Figure 35 illustrates the main screen of PolAnt. When the software is executed, the main screen of the GUI appears on the graphic environment of the operating system, allowing the user to interact with the algorithm with only mouse clicks.

![Screenshot of the PolAnt GUI main screen](image)

PolAnt was also designed to integrate the PEP functions of the Policy-Based overlay network management system. It is capable to modify the parameters of the overlay simulation during runtime when the petitions are received from the PDP.

4.4.4.1 PolAnt as a Graphical User Interface for BlatAnt

The original command line interface of BlatAnt was analyzed looking for improvements. The BlatAnt algorithm was designed to run entirely from the command line or console. When the main program is executed without arguments as illustrated in Table 3, the full list of commands and parameters is displayed, and next to each parameter, a help text with the explanation of the function of the commands or parameters is shown. The default values for each parameter are also displayed. All the parameters of the algorithm can be modified from the command line interface (CLI). If the parameter is not called, the algorithm uses it with its default value.

```
Usage: Short format, Long format, <Type>, Description -- (Mandatory) [Default value]

-h --help            <n/a>    Show this help -- (optional) [false]
--info             <n/a>    Show copyright info and redistribution license -- (optional) [false]
-f --fileinterval <integer> File output interval -- (optional) [10]
--screeninterval  <integer> Screen output interval -- (optional) [20]
--maxiter          <integer> Maximum iterations -- (optional) [2500]
--cpufactor        <integer> Threads / Cores ratio -- (optional) [1]
```
Table 3: Original user interface for the BlatAnt algorithm

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--initial</td>
<td>&lt;string&gt;</td>
<td>Initial graph filename -- (mandatory)</td>
</tr>
<tr>
<td>--prefix</td>
<td>&lt;string&gt;</td>
<td>Output prefix -- (mandatory)</td>
</tr>
<tr>
<td>--alphamaxsize</td>
<td>&lt;integer&gt;</td>
<td>Alpha table maximum size -- (optional) [20]</td>
</tr>
<tr>
<td>--alphamaxage</td>
<td>&lt;integer&gt;</td>
<td>Alpha table maximum age -- (optional) [20]</td>
</tr>
<tr>
<td>--iv</td>
<td>&lt;integer&gt;</td>
<td>Ant vector length -- (optional) [15]</td>
</tr>
<tr>
<td>--kappa</td>
<td>&lt;float&gt;</td>
<td>Random walk probability -- (optional) [0.5]</td>
</tr>
<tr>
<td>--omega</td>
<td>&lt;integer&gt;</td>
<td>Update interval omega -- (optional) [15]</td>
</tr>
<tr>
<td>--omegac</td>
<td>&lt;integer&gt;</td>
<td>Update interval omega connection -- (optional) [15]</td>
</tr>
<tr>
<td>--omegad</td>
<td>&lt;integer&gt;</td>
<td>Update interval omega disconnection -- (optional)</td>
</tr>
<tr>
<td>--rho</td>
<td>&lt;float&gt;</td>
<td>Residual pheromone -- (optional) [0.5]</td>
</tr>
<tr>
<td>--psi</td>
<td>&lt;float&gt;</td>
<td>Default pheromone decay -- (optional) [0.9]</td>
</tr>
<tr>
<td>--omega</td>
<td>&lt;integer&gt;</td>
<td>Default pheromone increment -- (optional) [0.1]</td>
</tr>
<tr>
<td>--epsilon</td>
<td>&lt;float&gt;</td>
<td>Minimum pheromone -- (optional) [0.005]</td>
</tr>
<tr>
<td>--initpheromone</td>
<td>&lt;float&gt;</td>
<td>Initial pheromone -- (optional) [1.0]</td>
</tr>
<tr>
<td>--betadecay</td>
<td>&lt;float&gt;</td>
<td>Beta pheromone decay -- (optional) [0.9]</td>
</tr>
<tr>
<td>--gammadecay</td>
<td>&lt;float&gt;</td>
<td>Gamma pheromone decay -- (optional) [0.5]</td>
</tr>
<tr>
<td>--deltadecay</td>
<td>&lt;float&gt;</td>
<td>Delta pheromone decay -- (optional) [0.9]</td>
</tr>
<tr>
<td>--d</td>
<td>&lt;integer&gt;</td>
<td>D Parameter -- (optional) [6]</td>
</tr>
<tr>
<td>--k</td>
<td>&lt;integer&gt;</td>
<td>K Parameter -- (optional) [1]</td>
</tr>
<tr>
<td>--s</td>
<td>&lt;integer&gt;</td>
<td>S Parameter -- (optional) [1]</td>
</tr>
<tr>
<td>--highclustering</td>
<td>&lt;n/a&gt;</td>
<td>High clustering -- (optional) [false]</td>
</tr>
<tr>
<td>--dynnet</td>
<td>&lt;n/a&gt;</td>
<td>Dynamic Network -- (optional) [false]</td>
</tr>
<tr>
<td>--freezing</td>
<td>&lt;n/a&gt;</td>
<td>Enable freezing -- (optional) [false]</td>
</tr>
<tr>
<td>--antage</td>
<td>&lt;integer&gt;</td>
<td>Ant maximum age -- (optional) [100000000]</td>
</tr>
<tr>
<td>--birthprob</td>
<td>&lt;float&gt;</td>
<td>Ant birth probability -- (optional) [0.15]</td>
</tr>
<tr>
<td>--iota</td>
<td>&lt;integer&gt;</td>
<td>Ant respawn interval -- (optional) [0]</td>
</tr>
<tr>
<td>--addchainint</td>
<td>&lt;integer&gt;</td>
<td>Add chain interval -- (optional) [250]</td>
</tr>
<tr>
<td>--remchainint</td>
<td>&lt;integer&gt;</td>
<td>Remove chain interval -- (optional) [50]</td>
</tr>
<tr>
<td>--addchainlen</td>
<td>&lt;integer&gt;</td>
<td>Add chain length -- (optional) [25]</td>
</tr>
<tr>
<td>--freeride</td>
<td>&lt;n/a&gt;</td>
<td>Free ride -- (optional) [false]</td>
</tr>
<tr>
<td>--freerideruns</td>
<td>&lt;integer&gt;</td>
<td>Free ride runs -- (optional) [10]</td>
</tr>
<tr>
<td>--seed</td>
<td>&lt;integer&gt;</td>
<td>Random Seed -- (optional) [81]</td>
</tr>
<tr>
<td>--balancing</td>
<td>&lt;n/a&gt;</td>
<td>Balancing algorithm -- (optional) [false]</td>
</tr>
<tr>
<td>--balhops</td>
<td>&lt;integer&gt;</td>
<td>Balancing hops cycle length -- (optional) [10]</td>
</tr>
<tr>
<td>--disableblatant</td>
<td>&lt;n/a&gt;</td>
<td>Disable the Blåtant Algorithm -- (optional) [false]</td>
</tr>
</tbody>
</table>

The main effort in the PolAnt GUI was to redesign the user interface into a graphical, intuitive, user friendly and self-explanatory user interface and integrating it with the BlatAnt algorithm simulator.

A screen capture of the graphical user interface of PolAnt can be seen in Figure 36. The design of the interface and menus was carefully selected to be user friendly and intuitive for the user. The look and feel of the main window is adapted automatically to the current operating system look and feel. The parameters were grouped into tabs, and throughout the whole interface, each component or parameter is explained to the user by popup captions with a clear statement when the mouse pointer is over each element. Furthermore, new features were included into the graphical interface that simplify the interpretation of the results and also, the usability of the user interface was improved by storing data about the different simulations that the command line version was unable to store.
The initial version of the BlatAnt implementation does not restrict invalid parameters in the CLI, we therefore implemented methods for restricting the input of the parameters to only the valid ones. Furthermore, contrary to the initial version, the user is not able to run the algorithm until the minimum set of required parameters has been set avoiding software exception faults. Also, as illustrated in Figure 37, for selecting the input files and saving the result files, a browse button, which calls the file navigation component, was added to make easier the selection of files to the user. Furthermore, when a file has been selected, the path of this file is stored in a text field so the user can access easily the files by copying the text.

As an added feature, there is an option available in the view menu, as illustrated in Figure 38 for showing the window with statistics from a specific result file and when the simulation has finished, another option to display the statistics from the last iteration. The Swing menu item calls a Python program which performs calculations in the result files using the NetworkX libraries of Python, and then returns the results of the statistic calculations in a window on
the GUI. The statistics shown can be found in Table 4, which shows an example of the statistics from an output file.

| Radius: 25 |
| Nodes: 50  |
| Edges: 98  |
| Diameter: 49 |
| Maximum Degree: 2 |
| Minimum Degree: 1 |
| Average Degree: 1 |
| Degree Variance: 0.96 |
| Standard Deviation Degree: 0.979795897113 |

Table 4: Statistics from result file

This statistics shows the average, variance and standard deviation of some of the graph file provided as input. The results displayed are: radius of the graph (maximum graph distance from the center of the graph to a node in the edge), number of nodes and edges, diameter of the graph (maximum distance between two nodes), and degree (number of neighbors of a node).

Another added feature was to display in a graphical manner, the different result files that the algorithm produces. In the view menu of the GUI, a menu item is added to represent a visualization of the network graph. The option can be used also for input files. The screenshot in Figure 39 shows an example of the visualization of a bus topology after 80 iterations of the algorithm.

This representation of the overlay network is useful to observe graphically the evolution in the creation of the overlay links in the network. Is interesting to see how by enforcing the
policies, the modifications of the different parameters affect in different ways to the structure and connections among the overlay nodes.

### 4.4.4.2 PolAnt as a Policy Enforcement Point

The main goal of PolAnt is to act as a PEP. When the user has started to run the simulation of the algorithm, the software runs a routine in the background that waits for data which arrive over a socket. However, the software will only apply the policy, modifying the parameters of the current simulation when the set of conditions requested in the policy match the conditions requested by the PDP.

When a new connection has been received, the PDP is asking PolAnt for a value of the specific parameter of the condition part of the policy. PolAnt will reply with the value asked. If the value meets the condition in the policy, in a second petition, the actions to enforce are requested. The current parameters are enforced and the policy is applied. The latter is responsible for showing in the text area of the program the current parameters before applying the policy, and the actions enforced with the policy. Figure 40 shows the reaction of the software when the policy is received and the condition is satisfied. Note that the value for the *birth probability* changed from 0.3 to 0.1.

![Figure 40: PolAnt when policy is applied](image)

When the policy is received, but the condition is not satisfied, the software shows the actual parameters in the text area, to indicate that the policy has been received, but not applied.
5. Policy-Based management of the BlatAnt overlay network

5.1. Overview

As described in the previous section, the software developed reacts automatically to user-defined rules according to changes of context information in the underlying network. We present in this section the results of an adaptive behavior of BlatAnt in the construction and maintenance of the overlay network according to the rules of our Policy-Based Network Management System implemented for this purpose. With this extension, the overlay network that the BlatAnt algorithm creates, is capable to adapt optimally and automatically to the context changes in a way which is simple for the network administrator, and also flexible and extensible.

As explained in the previous sections, nodes can dynamically join and leave the overlay network. Besides, the overlay network is not aware in real time of the changes on the underlying network (a part of the network has been disconnected) and also, the context activity on the underlying network (situations of high network activity, overload, etc.). The result of BlatAnt without a Policy-Based Network Management System can be a non-optimal functioning of both the underlying and the overlay network. With the introduction of the elements of the Policy-Based Network Management scheme, the overlay network is to rapidly react and optimally maintain a good performance in the underlying network, and an optimal management of the structure and behavior of the overlay network.

Despite the actions performed on the overlay network to reconfigure and modify the overlay structure, the context information by the context generator for the different scenarios is only triggered in the Policy-based system to reconfigure the network but the context information does not change the overlay structure. i.e. the overlay structure is modified by the Policy-based system but it does not suffer the dynamic changes when they are triggered. The reason of it is that the BlatAnt simulator in the version used for this work does not support the simulation of dynamic changes in the network in a straightforward manner. The support of dynamic changes would require further study and development in a different research direction. However, as the focus of this thesis was to develop a Policy-based system, this fact did not impede the study of the reaction of the overlay network structure and of the behavior of the algorithm.

5.2. Default policies

The policies defined and explained in Section 3.4 are formally defined in Table 5. We have chosen these policies based on the sensitivity analysis results of the BlatAnt algorithm, and with the characteristics of this algorithm in mind, we have thought of different situations that can occur on the underlying network and how BlatAnt should modify its parameters to react to the current situation in the network. The enforcement of these policies was tested and proved to adapt the construction of the overlay network to changes of the context in the underlying network. Our system adapts the creation or maintenance of the overlay network according to the policies defined and based on the occurrence of events.
Besides the policies detailed in Table 5, the network administrator can create new policies by just defining them in the PMT. To define a new policy in the PMT, the user needs to know that the field “when” represents the situation that needs to occur in the network, the “if” field represents the condition that the current settings of the algorithm must have (i.e. status of a running parameter), and the “execute” field are the set of parameters to modify in the current simulation. New policies will be added to the list of existing policies, and every new policy that satisfies the condition upon the occurrence of a certain event will be enforced in the system.

<table>
<thead>
<tr>
<th>Policy Name</th>
<th>when (event)</th>
<th>if (condition)</th>
<th>execute (action)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regeneration</td>
<td>Network Split</td>
<td>Birthprob. ≤ 0.15</td>
<td>Birthprob. = 0.2</td>
</tr>
<tr>
<td>Reduce Connections</td>
<td>Very high number of overlay links</td>
<td>Dparameter ≤ 6</td>
<td>Dparameter = 8</td>
</tr>
<tr>
<td>Stability</td>
<td>Overlay links are stable</td>
<td>Alphatable ≤ 20</td>
<td>Alphatable = 25</td>
</tr>
<tr>
<td>High Robustness</td>
<td>Attacks</td>
<td>VectorLen. ≤ 15</td>
<td>VectorLen. = 19</td>
</tr>
</tbody>
</table>

Table 5: Formal definition of Policies

Once policies such as those described above are defined, their actions are automatically enforced by the management system with the purpose of re-configuring the overlay network in response to the defined network conditions.

5.3. Analysis of the Policy-Based experiments

We analyze in the effects of the policies formally defined on the previous section on the overlay network. The reason for these experiments is twofold: firstly, to evaluate the validity of the policies defined and secondly, to test the correct operation of the developed system. The results of the experiments are shown in charts where we can see relevant data from the overlay network topology, such as network diameter and number of edges across the different iterations. The triggering of the management policies are indicated with a vertical dash line, and after this line, the enforcement of the policies in the current simulation is performed and therefore the desired effects on the overlay network structure are appreciated.

The results of the sensitivity analysis of the modification of a single parameter during runtime shown in Section 3.5 should show approximately the same effects than the results obtained when single policies are enforced if the policy triggered is affecting the same parameter. In the following section, policies are enforced automatically when a change in the context information is executed. We also ran the simulation with the different parameters and the values selected for the policies are also observed in the sensitivity analysis. The only reason for small variations is the number of the iterations after which the policy is enforced, and also small variations in the behavior of the algorithm can be observed when we run different simulations even when affecting the same parameters.
5.4. **Single policy applied**

We ran a longer simulation than in the previous sensitivity analysis, with 1200 iterations. The simulation is started with the default values for all its parameters and a single policy is enforced in each simulation from the 400\textsuperscript{th} iteration and onwards. The reaction of the overlay network and the desired effect of the policies are discussed in each subsection.

5.4.1 **Policy: Regeneration**

The aim of this management policy is to restructure the overlay network when a part of the network has been disconnected from the whole network due to a network split. The paths that the nodes use to communicate between each other node in the network would need to be restructured choosing new routes.

Figure 41 and Figure 42 show how after the policy is applied in the 400\textsuperscript{th} iteration the desired effect can be seen in the overlay topology: the diameter in the network is not affected, but the increase in the edges is highly noticeable. This increase of edges is produced due the creation of new discovery ants, and these ants will create new links between the overlay hosts that will likely lead to alternative routes between the nodes. These routes are probably improving the performance when a network split break some paths and makes the connection between nodes unavailable.

![Figure 41: Evolution of diameter when the policy "Regeneration" is applied](image-url)
5.4.2 Policy: Reduce connections

The purpose of this policy is to reduce the creation of edges on the graph, which is translated into reduction of new links in the overlay network and, in case that the overlay network has not yet reached the upper bound of diameter indicated for the $D$ value, the policy would also bring stabilization or even an increase of the diameter of the network. The reason of the increased diameter is that the upper bound would increase, and this could make the diameter increase. The reason of the stabilization in the creation of edges is because once the network has already reached the expected diameter, less edges would be necessary to maintain the diameter of the network or achieve the expected diameter.

A possible scenario to apply this policy could be that the computation in the nodes is very high, and therefore, the complexity of the management to maintain the network diameter is increased. An increment of the diameter is desired in this case. By applying this policy the two mentioned goals are obtained. Figure 43 and Figure 44 show the effect of the reduction of edges in the overlay topology after the policy is applied in the 400th iteration, but because the network has already reached the expected diameter, the diameter in the graph after applying the policy is slightly affected, slightly increasing in some moments compared to the expected diameter. As soon as the policy is applied, the creation of new links is stabilized and therefore, the number of edges is almost constant. By examining the graph in the last iterations Figure 44 in it seems that the number of edges tends to be reduced.
5.4.3 Policy: Stability

The main goal of this policy is to adapt the overlay to a situation of stability in the network. For example, if the nodes are static and there is no detection of nodes joining or leaving the network, this policy will reduce effectively the number of edges while still reducing the
diameter. With this policy, the *alpha table maximum size* of each node is increased, and therefore each node can store more information about the rest of the nodes in the overlay. This information can be useful when the situation of stability appears in the network. As discussed in section 3.2.1, a reduction in the number of edges is observed due to more information being available to the nodes. Therefore, more probability that the new creation of overlay links will lead to optimal paths, the overlay links created do not need to be removed, and then fewer edges are generated.

However, when applying this policy, it needs to be considered that the computation on each node would be increased, and in this case the value of the birth probability defined in the policy should be limited to take into account the computational capacity of each node. Figure 45 and Figure 46 are showing the results after applying this policy. It can be observed how the number of edges does not increment after the policy of resilience has been employed after the 400<sup>th</sup> iteration.

![Figure 45: Evolution of diameter when the policy "Resilience" is applied](image)
5.4.4 Policy: High Robustness

The objective of this policy is to create a robust network topology to overcome a possible scenario of a critical node crashing (a central node with many overlay links to neighbors and many routes are depending on this node) or to overcome the issue of certain number of central nodes leaving the network. This scenario could be produced, for example by a situation of network attacks in a period of time. This policy can also react to a situation of high instability of network resources. For example, nodes that switch from an offline to an online state in the underlying network, or links that are temporarily unavailable.

Figure 47 and Figure 48 show how the policy is applied after the 400th iteration. The effect of an exponential increase of edges can be easily seen on Figure 48. This exponential increase of edges could be translated into a more robust topology, since the increase of edges in the topology keeping the diameter at the same level, is translated into more alternative routes between nodes. In the case of a crash of several nodes or if a critical node crashes, the alternative routes could be used as backup routes for the ones that are not available.
5.5. Multiple policies

After analyzing the behavior of the algorithm with single policies in a long simulation, we decided to see the results with even on a longer simulation period, this time with 1900
iterations. The behavior of the algorithm was analyzed when different policies were enforced sequentially.

We did not perform an exhaustive test with all the possible combinations of parameters and values, as these experiments would require a considerable amount of time and computation power. However, the experiments that are shown are representative enough to prove that in a situation where different situations occur in the network context at the same time in the discussed scenarios, different policies can be applied to achieve multiple goals. Despite the fact that the results observed fit the patterns presented in the previous section, it is nevertheless important to say as we mentioned in Section 3.4, that the modification of several parameters during runtime would require a more exhaustive study to assess the behavior of the algorithm.

5.5.1 Robustness & Stability

In the following experiment we performed the same simulation that was performed for evaluating the policy “Robustness” mentioned in the previous section. But at the 1000th iteration, when the network topology seems to be at the most robust level, with the higher number of edges, the policy “Stability” was triggered. After the second policy is triggered, the effect of stability observed in the graphs shown in Figure 49 and Figure 50 seems to have the same behavior than when the single policy was applied over the default parameters.

![Figure 49: Evolution of diameter after applying Robustness at 400th and Stability at 1000th iteration](image-url)
5.5.2 Robustness & Reduce connections

Because both policies “Resilience” and “Reduce connections” produce a decrease in the creation of edges, this time, after applying the policy “Robustness”, the policy “Reduce connections” was chosen to be triggered, as we wanted to compare the difference on the reduction of the edges after applying the first policy, and also, to check how the combination of a larger value of the optimization parameter $D$ would affect the algorithm when considered in conjunction with a larger value for vector length. The results shown in Figure 51 and Figure 52 indicate that the second policy does not have apparently any effect on the overlay topology. That is probably because while the overlay network has already reached the expected diameter, the robustness policy is still active, and the number of edges is still growing, at the same time that the second policy is stopping the increase in the number of edges.
5.6. Discussion

The realized experiments show once more that there is indeed a difference on the network topology structure, and in the creation and maintenance of the overlay network depending on the parameters of the algorithm during runtime. The results have also shown that the
different pieces of software and the confection of policies worked well according to the high level objectives of the policies in the different scenarios discussed.

The results of the experiments shown can help the network administrator of the overlay network that BlatAnt creates with an understanding of the overlay network that BlatAnt creates, and how the behavior of the algorithm is affected when the different policies are triggered. The results shown can help to decide which network metrics are useful to monitor, which parameters are useful to adapt, and what policies are useful to enforce in actual implementations of the BlatAnt overlay network. The administrator can also have a theoretical point of view of which parameters can be tweaked, in order to adapt the overlay network for different types of underlying networks, and which are the parameters that help to improve the performance of the algorithm, for different types of network contexts.
6. Conclusion

Overlay networks offer an effective way to manage highly distributed scenarios, for example mobile ad-hoc scenarios, multilayer and heterogeneous networks. Distributed communication, computing and storage operate more efficiently with the use of network overlays. However, in scenarios with frequent volatility of resources or mobility, the performance of the overlay network elements can be affected and therefore, adaptation on such dynamic situations needs to be performed. Furthermore, the increased complexity of manual management creates a need of an automatic system that allows the prevention of future situations. The two aforementioned problems can be solved with the use of a Policy-Based overlay network management system. With this approach, the administrator can define network-wide configuration rules to overcome possible problems or new application requirements that allow for adapting to evolutionary changes in the overlay network. This work is a contribution to a wide project where the management supports the dynamic requirements of different applications running on higher levels. The evaluation of Policy-Based Network Management in Bio-Inspired overlay networks was assessed in this thesis, addressing performance and self-management viability for different scenarios.

The purpose of this work was to create a solution that could efficiently manage the overlay network created for the SmartGRID project [9], developed within the Pervasive and Artificial Intelligence (PAI) research group at the University of Fribourg. In this matter, a Policy-based overlay network management system was developed. Our solution allows a dynamic and automatic management of the overlay network that the BlatAnt [8] algorithm creates, without modifying the implementation of the algorithm.

Policy-based network management system (PBONMS) was selected as a management approach because the architecture is designed in a hierarchically manner, allowing the integration in existing overlay networks without modifying the structure or operation. We adopted a hierarchical design for a possible in an effort support future new functionalities and make easier implementation of the PBONMS in a physical overlay network. Additionally, the research and methodology followed in this thesis could be again implemented for the case of a different network algorithm in the assessment of a Policy-Based system, and the definition of policies. Furthermore, the effects desired in the designed policies were validated, proving that the overlay network can be reconfigured efficiently, adapting its structure to the potential changes in the context for an optimal functioning of the network overlay.

A subject discussed in this thesis is the issue of removing the network administrator out of the management loop. The experiments done during this research showed us that when the underlying structure and requirements of the overlay network change, previously defined policies could not make sense if substantial changes occur in the underlying network. It would require further study whether this situation could be solved with the automatic creation and adaptation of policies, by learning from the information that the context provides over time. Furthermore, defining high level goals to manage present and future network conditions, maintaining the benefits after this evolution suppose a challenge for a
completely autonomic system and it would require the addition of new elements, adaptations in the Policy-based network management scheme and further research [39] [46]. Therefore, we can state that, the network administrator cannot be removed from the management loop as it can be a moment in a future time where the policies defined in a present time are no longer valid due to important changes in the underlying network. The policies need to evolve according to the usage and requirements of the managed network. At the moment these kinds of decisions can only be made by the network administrator.

In the future, this work could lead to the automatic generation of policies by learning from the context behavior over time, supporting also the dynamic changes in the network simulator. Additionally, a completely distributed solution where the elements of the Policy-Based Overlay Network management system are not fixed, and such components are able to change the location in a coordinated way should be considered as the nature of the overlay network is shifting everyday to more mobile and dynamic scenarios.

Finally, it is noteworthy to mention that the study of bio-inspired approaches gave the author of this thesis an insight about how much we can learn about the nature, and the examples studied spurred an interest for researching about different mechanisms or approaches that the nature and especially the biology (e.g. genetic virus mutation for non-detection, host-parasite coevolution) has in common with the computer science world.
7. References


8. Annex

In the annex we can find the source code of all the developed software elements, and also the necessary information on how to run the programs. A CD-ROM is provided together with this Master Thesis with the code and the instruction for generating the database, and the database tables and elements.

8.1. Source code (CD-ROM)

In the CD-ROM we can find the source code and binaries of the software developed. There is a directory for each software component. Inside each directory we can find the source code (in the directory: src) and the executable JAR files (in the directory: dist). The script for the creation of the database is in the root directory. Each software component can be imported to Netbeans or any other framework, to perform modifications in the code.

8.2. Database Information (CD-ROM).

The database is written in SQL script. When this SQL script is executed, the different databases and tables will be created and the data that define the different policies will be added to the database. For the correct communication of the database with the policy editor, the database user was written for the user “root” with the password “1234”.

```
CREATE DATABASE Polant;
USE Polant;

CREATE TABLE IF NOT EXISTS `rules` (
  `name` varchar(20) NOT NULL,
  `dparameter` int(11) NOT NULL,
  `alphasize` int(11) NOT NULL,
  `vectorlength` int(11) NOT NULL,
  `birthprobability` double NOT NULL,
  `ifx` varchar(20) NOT NULL,
  `whenx` varchar(20) NOT NULL,
  PRIMARY KEY (`name`
) ENGINE=MyISAM DEFAULT CHARSET=latin1 COMMENT='policies';

INSERT INTO `rules` (`name`, `dparameter`, `alphasize`, `vectorlength`, `birthprobability`, `ifx`, `whenx`) VALUES
('Network Split', 0, 0, 0, 0.2, 'Birthprob.<=0.15', 'High nr. of Packets'),
('Reduce Connections', 8, 0, 0, 0, 'Dparameter<=6', 'High nr. of Edges'),
('High robustness', 0, 0, 19, 0, 'VectorLen.<=15', 'Links are dynamic'),
('Stability', 0, 25, 0, 0, 'Alphatable<=20', 'Links are stable');
```

8.3. Complete set of result graphs for the sensitivity analysis

In this section, the full list of charts that were produced in the context of the sensitivity analysis is shown. These charts show the behavior of the algorithm in a bus, hypercube and grid topology.
grid-32x32

- BirthProbability=0.10
- BirthProbability=0.15
- BirthProbability=0.20

Diameter vs. Iterations

grid-32x32

- BirthProbability=0.10
- BirthProbability=0.15
- BirthProbability=0.20

Edges vs. Iterations

2D-1 = 11
grid-32x32

- Diameter
- Iterations

D=5
D=6
D=7

2D-1 = 13
2D-1=11
2D-1=9

grid-32x32

- Edges
- Iterations

D=5
D=6
D=7
grid-32x32

Diameter vs. Iterations

VectorLength=11
VectorLength=15
VectorLength=19

2D-1 = 11

Edges vs. Iterations

VectorLength=11
VectorLength=15
VectorLength=19
The graphs show the diameter and edges for different values of $\text{AlphaMaxSize}$ in the `hyper-1024` network.

- **Diameter**
  - $2D-1 = 11$
  - 4 iterations are shown: $\text{AlphaMaxSize}=15$, $\text{AlphaMaxSize}=20$, $\text{AlphaMaxSize}=25$

- **Edges**
  - 4 iterations are shown for each value of $\text{AlphaMaxSize}$.
The diagrams illustrate the behavior of different overlay networks under varying birth probabilities.

The upper diagram shows the diameter (2D-1) of the network as a function of the number of iterations. The network is labeled as 'hyper-1024' and the birth probability is varied. The diameter is measured as a horizontal distance from the horizontal axis.

The lower diagram depicts the number of edges in the network as a function of the number of iterations. Again, the network is labeled as 'hyper-1024' and the birth probability is varied. The number of edges is measured vertically from the horizontal axis.
### path-1024

**Diameter**

- $\cdots\cdots\cdot$ AlphaMaxSize=15
- AlphaMaxSize=20
- $\cdots\cdots\cdot$ AlphaMaxSize=25

**Iterations**

<table>
<thead>
<tr>
<th>Iterations</th>
<th>10</th>
<th>110</th>
<th>210</th>
<th>310</th>
<th>410</th>
<th>510</th>
<th>610</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter</td>
<td>0</td>
<td>204</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Edges**

- $\cdots\cdots\cdot$ AlphaMaxSize=15
- AlphaMaxSize=20
- $\cdots\cdots\cdot$ AlphaMaxSize=25

**Iterations**

<table>
<thead>
<tr>
<th>Iterations</th>
<th>10</th>
<th>110</th>
<th>210</th>
<th>310</th>
<th>410</th>
<th>510</th>
<th>610</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edges</td>
<td>0</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
path-1024

Diameter

Iterations

D=5
D=6
D=7

2D-1 = 13
2D-1 = 11
2D-1 = 9

path-1024

Edges

Iterations

D=5
D=6
D=7