PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 05/13

A Self-Organizing and Normative Piloting System

Manoel Teixeira de Abreu Netto
Carlos José Pereira de Lucena
Baldoino Fonseca dos Santos Neto
Elder José Cirilo Reioli
Firmo Freire

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 05/13 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena April, 2013

A Self-Organizing and Normative Piloting System *

Manoel Teixeira de Abreu Netto, Carlos José Pereira de Lucena, Baldoino Fonseca dos Santos Neto, Elder José Cirilo Reioli, Firmo Freire

- mnetto@inf.puc-rio.br, lucena@inf.puc-rio.br, bneto@inf.puc-rio.br, ereioli@inf.puc-rio.br, firmo@les.inf.puc-rio.br

Abstract. The services and technologies inherent to computer networks have become part of society. However, its management by human administrators came at high cost and it is prone to failure, and the simple automation of management through software components may worsen the situation due to the wide variety of systems and unexpected behaviors. Autonomic networks were proposed to deal with this management problem by enabling systems to self-manage. But, in order to perform a self-management in an optimal, robust and secure way it is necessary to have a piloting system. The main goal of a piloting system is to regulate and adapt the virtual network in response to changing context in accordance with applicable high-level goals and policies. In this context, this report presents a self-organizing and normative piloting system that aims to govern the entities of the network in a decentralized way. Moreover, we provide a simulation environment that enable users to experiment and observe the network behavior in face of the application of different normative and organizational configurations of the piloting plane.

Keywords: Network Virtualization, Self-Organization, Multi-agents System.

---

* This work has been sponsored by the FAPERJ.

Monografias em Ciência da Computação, No. 05/13 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Abril, 2013

A Self-Organizing and Normative Piloting System *

Manoel Teixeira de Abreu Netto, Carlos José Pereira de Lucena, Baldoino Fonseca dos Santos Neto, Elder José Cirilo Reioli, Firmo Freire

- mnetto@inf.puc-rio.br, lucena@inf.puc-rio.br, bneto@inf.puc-rio.br, ereioli@inf.puc-rio.br, firmo@les.inf.puc-rio.br

Resumo. Os serviços e tecnologias inerentes às redes de computadores já se tornaram parte da sociedade. Entretanto, seu gerenciamento por administradores humanos vem a um alto custo e suscetível a falhas, ademais, a simples automação deste gerenciamento através de componentes de software pode piorar a situação por conta da extensa variedade de sistemas e comportamentos inesperados. Para lidar com esses problemas uma das soluções propostas são as Redes Autonômicas, que permitem um autogerenciamento do sistema. Porém, para realizar essa tarefa de forma otimizada, robusta e segura é necessário possuir um sistema de pilotagem. O principal objetivo de um sistema de pilotagem é regular e adaptar a rede virtual de forma que esta possa responder às mudanças de contexto de acordo com políticas e objetivos de alto nível. Neste cenário, este relatório apresenta um sistema de pilotagem normativo e auto-organizável que atua no governo e gerenciamento das entidades da rede virtual numa abordagem descentralizada. Ademais, provemos um ambiente de simulação que permite usuários experimentarem e observarem o comportamento da rede virtual em face à aplicação de diferentes configurações de normas e de organizações do plano de pilotagem.

* Este trabalho foi financiado pela FAPERJ.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

Introduction

Computer networks have become part of society. Its services and technology have multiplied and they have become essential to the global economy. Its management by human administrators, however, became costly and prone to failure, and the simple automation of management through software components may worsen the problem due to the wide variety of systems and unexpected behaviors. Autonomic networks were proposed to deal with this management problem. They represent a specific topic in the area of autonomic computing, a term coined by IBM, intended to deal with complexity by enabling systems to self-manage. Today, it is also advocated the approach of plural-ism of architectures for the future Internet over the one-size-fits-all TCP/IP [Turner and Taylor, 2005]. This new approach defines that network providers should be splitted in service and infrastructure providers [Feamster et. al., 2007] and proposes the use of virtualization [Anderson et al., 2005]. Users request network services from the service providers, which instantiate virtual networks over the substrate provided by the infrastructure providers. Each virtual network can have its own protocols and configurations, in accordance with the objectives of the service running on it, and must have isolation, i.e., the operation of virtual networks does not cause interference between them, although they are on the same infrastructure.

To achieve these objectives in an optimal, robust and secure way it is necessary to have a piloting system. A piloting system, used to control and manage the virtual networks, can be seen as an aggregation of two specific planes: a knowledge plane and an orchestration plane. The knowledge plane is in charge of recovering the knowledge useful for feeding the control and management algorithms. The orchestration plane, on the other hand, is in charge of indicating the course of the virtual network. The advantage of the piloting system is the possibility to adapt in real time through the management and orchestration plane. The piloting process aims to adapt the virtual network to new conditions and to take advantage of the intelligent decisions to alleviate the global network. Therefore, the role of the piloting system is to govern and adapt the virtual network in response to changing context in accordance with applicable high-level goals and policies. It supervises and integrates all other planes' behavior, ensuring integrity of management and control operations. In this context, the use of a multi-agent system permits the achievement of a more attractive orchestration process due to the following points: (1) each agent holds different processes (dynamic planners, low coupling); (2) the agents are cooperative and reactive, in the sense that they are able to use a privileged view of their neighbors and individual knowledge together.

As mentioned, the purpose of the piloting system described in this report is to regulate and integrate the behaviors of the network in response to changing context and in accordance with applicable high-level norms. Norm is a regulation mechanism that defines a set of rules to the system agents in order to ensure a social order that enables the achievement of the global goal of the organization. Our piloting system can be seeing as a self-organizing control framework into which any number of network devices can be plugged into or out of in order to achieve the required service level agreement. Therefore it hosts several self-organizing piloting systems each one managed by a piloting agent. Each agent maintains its own knowledge base consisting of a set of data models about the physical and virtual devices. In this way, agents manage a virtual devices by following a set of norms and using a set of knowledge. Moreover, agents can communicate and cooperate with each other by using behaviors.

In addition to the piloting system, we have implemented a simulation environment which allows users to experiment with agent-oriented pilot plans. Our simulation environment integrates an user interface with a normative multi-agent system and a virtualization environment. This environment was implemented in Java and the Gingko [Ginkgo, 2008] platform that supports the implementation of the multi-agent system. We are using the OpenFlow [McKeown et al., 2008] system to simulate the virtual networks. Moreover, we have developed a programming interface that allows the integration of the simulation environment with any virtualization environment.

Background

Multiagent Systems

Multi-agent systems [Ferber, 1999] are composed of intelligent entities, called agents, that have the capabilities needed to make the network autonomic. As shown in [Ferber, 1999]: (1) they are able to communicate, (2) possess their own resources, (3) perceive their environment, (4) have a partial representation of their environment, (5) have a behavior which aims at realizing their goals.

Thanks to such properties, multiagent systems can constitute a good tool to provide the autonomic scheme by guaranteeing the different characteristics which seem necessary to reach an autonomic behavior. In the following, we describe in more detail multiagent systems characteristics:

- Decentralization: The multi- agent approach is decentralized by definition and this decentralization aims at overcoming the incapacity of the classic Artificial Intelligence to operate in the current systems that are more and more distributed and decentralized. No agent possesses a global vision of the system and the decisions are taken in a totally decentralized way;
- Reactivity: One of the basic attributes of an agent is to be situated (situatedness, [Brooks, 1985]). That is, an agent is a part of an environment and it reacts according to what it perceives of this environment. The reactivity characteristic is very important in a context of highly dynamic networks, in which the decisions have to suit current conditions.;
- Proactivity: The agent is capable of setting goals and realizing them by executing plans, interacting with other agents, etc. In this case, the agent has more knowledge of its capabilities and on those of the other agents and is able to set up a strategy allowing it to evolve in its environment and to reach its objectives;
- Sociability: The multi- agent approach provides the ability to distribute the intelligence among different agents composing the system. This implies that an agent can handle some tasks individually but cannot do everything by itself. It needs to cooperate with the other agents and to rely on their help to get better results;
- Adaptability: In order to provide more flexibility, researchers are interested in using learning techniques (e. g., genetic algorithm [Berger and Rosenschein, 2004], reinforcement learning [Dutta et al., 2005], etc.) to face unexpected situations. If we return to the autonomic networks, using agents having learning

capacity can be very beneficial and allows for a more effective adaptation to the evolutions of the networking domain.

Norms

According to [Tuomela, 1995] a norm has a general structure of a group of agents. Thus, a norm consists of four components: the addressees' agents, the action to be performed by them and, finally, the circumstances under which the action must be carried out [Tuomela and Bonnevier-Toumela, 1992], [Tuomela, 1995], [Tuomela and Bonnevier-Toumela]. Moreover, [Tuomela, 1995] classifies norms as one of two kinds: rules or (r-norms), and social norms or (s-norms).

Rules represent explicit agreements among agents, and are created by an authority. Rules are subdivided into two further classes as follows. Formal rules are those that include legal sanctions such as laws and regulations, and informal rules that are not in written form but communicated orally and include informal sanctions.

Social norms are norms accepted not through agreement but through mutual beliefs, and are also divided into two classes: conventions, which concern the whole society or social class and have social sanctions, such as approval or disapproval; and group-specific norms, which concern a group of agents in a society.

[Tuomela, 1995] also explains the conditions under which either rules or social norms ought to be fulfilled by the members of a group; these conditions cause a norm to be enforced and can be described, as follows: (i) promulgation condition refers to the fact that norms must be issued by an authority; (ii) accessibility condition states that all members of the group acquire the belief that they ought to comply with the norm; (iii) if many members of the group fulfill the norm, or at least are disposed to do so, it is said that the pervasiveness condition is satisfied; (iv) the motivational condition is met when at least some members sometimes fulfill the norm because they believe it is true and that they ought to do so; (v) the sanction condition refers to the existence of social pressure against members that deviate from the norm, and ,finally, (vi) for a rule, the acceptance condition is the conjunction of the promulgation and accessibility conditions, whereas for a proper social norm to be accepted, only the accessibility condition is needed. Thus, contrary to rules, s-norms do not need to be issued by an authority, but they have to be recognized as norms for all the members in a group. In this work, we consider rules as mechanism to regulate the agent's behaviors.

Self-Organization

The approach of self-organizing systems has increased its relevance and is used to deal with complex domains. The use of this approach enables the development of decentralized systems that exhibit certain dynamicity and adaptability to couple with previously unknown perturbations [Serugendo et al., 2004]. According to principles of self-organization, each component of the autonomic system obtains and maintains only local information available in the environment, in a decentralized way and without any external control, being restricted only to local interactions. It is based on these interactions that the system exhibits it macroscopic behavior, which may be observed from a global point of view. The multiagent system paradigm has been considered a promising solution for the building of self-organized systems [Sycara, 1998]. According to [Serugendo et al., 2005] self-organization systems can be classified, as follows:

Strong self-organizing systems: are those where there is no explicit control, whether internal or external.

Weak self-organizing systems: are those where there is a re-organization through actions of center or planned internal control.

Additionally, the behaviors of a self-organization system can be characterized by the following properties (mandatory or optional):

- Absence of an explicit external control - This is a mandatory property that indicates that the system is autonomous, which defines change and that its organization is based exclusively on internal decisions and does not follow any external control to perform a (re-) organization. This property refers to the "self' of self-organization.
- Decentralized control. A self-organizing system can work under decentralized control. In this case, there is no internal central authority or centralized information flow. In this way, the access to global information is limited by local interactions, which are governed by simple rules. This property is generally not mandatory, as we can see it in natural self-organizing systems, such as the bees.
- Dynamic Operation - This mandatory property is associated the evolution of the system. Considering that the organization evolves independently of any external control, this property implies in the self-organization process.

Related Works

Telecommunications Network Management systems are a type of system that can be categorized as large, complex and unpredictable. Current research focuses on policy based management and autonomous systems, using a variety of languages and technologies. The four main Autonomic Network Management systems are ANEMA [Derbel et al., 2009], FOCALE [Jennings et al., 2007] and Pronto [Sheridan-Smith et al., 2006]. We describe, critique and compare them with our piloting approach.

In ANEMA, the high-level objectives of the human administrators and the users are captured and expressed in terms of Utility Function policies through a set of mechanisms. The Goal policies describe the high-level management directives needed to guide the network to achieve the previous utility functions. Finally, the 'behavioral' policies describe the behaviors that should be followed by network equipment to react to changes in their context and to achieve the given 'Goal' policies.

To demonstrate the capacities of the ANEMA architecture, they explained how it should be instantiated in a multiservice IP network and how the proposed utility-based analytical models were used. The results confirmed that the proposed model allows specifying the optimal feasible state. However the result state is still not the optimal one. They also implemented a simulator of the system and perform a set of simulations based on several proposed scenarios.

One problem to be solved in ANEMA is how the autonomic routers diffuse the information to others routers on the environment. Another issue that is not so clear is how deep is the coupling between the stock router and an autonomic one. The ANEMA focuses on the self-configuration and a little on the self-optimization aspect, thus is not a complete autonomic solution, as our piloting approach. Neither they use biological inspired solutions.

On the other hand, FOCALE, Foundation Observation Comparison Action Learn rEason, propose the use of information and ontological modeling to capture knowledge relating to network capabilities, environmental constraints, and business goals and policies, together with reasoning and learning techniques, to enhance and evolve this knowledge. Also, to deliver full autonomic network management capabilities FOCALE introduce decentralized processes and algorithms into the network infrastructure modeled on various biological processes found in the nature world. As ANEMA, FOCALE uses policy-based network management system, incorporating translation/code generation processes that automatically configure network elements in response to changing business goals and/or environment context.

FOCALE have as a base element an AME (Autonomic Management Element) which handles a managed resource, be it single device or network, that is the same idea of our piloting agent. Also, FOCALE is based on the MAPE control loop described by [Kephart and Chess, 2003], and used by our solution, the Monitor, Analyze, Plan and Execute loop, but reduced to a maintenance control loop and an adjustment control loop.

Finally, Pronto specifies a Policy-based service definition language to describe services and the system model through service definitions. The language merely allows those services to be described by a network engineer, but it is responsibility of the management system to use the policies within the service definition to construct and manage individual services.

Policies can also be applied to a pluggable and automated management software component known as a Domain Expert. These components transform policies at a high level of abstraction into corresponding lower-level policies. A QoS Domain Expert instance with dynamic behavior will be used if congestion is detected, the Domain Expert will modify the low-level policies to reduce the Committed Information Rate (CIR) of each service.

Basically, the Pronto solution specifies a domain specific policy language, which defines desired parameters of each network device. There isn't an autonomic module or agent for each network element, but a virtual device, which controls the configuration of the associated network device. Here, the term virtual device is not the same as we use in our piloting network. The authors didn't describe the whole architecture, so we don't know if it is centralized or distributed, or how the policies are diffused to others devices to couple with unpredictable situation. They do not use agents or biological inspired solution.

A Self-Organizing and Normative Piloting System

In order to provide autonomy to the virtual networks, we developed a multiagent self-organizing and normative piloting system. The idea is to adjust the network flow and routes in an autonomic way, without any explicit central control, maintaining the quality of service defined in the SLA and controlling the agents behaviors through norms. So, the virtual networks devices have a piloting agent responsible to capture and diffuse information among neighbors and act in the configuration and management of the router under a local perspective.

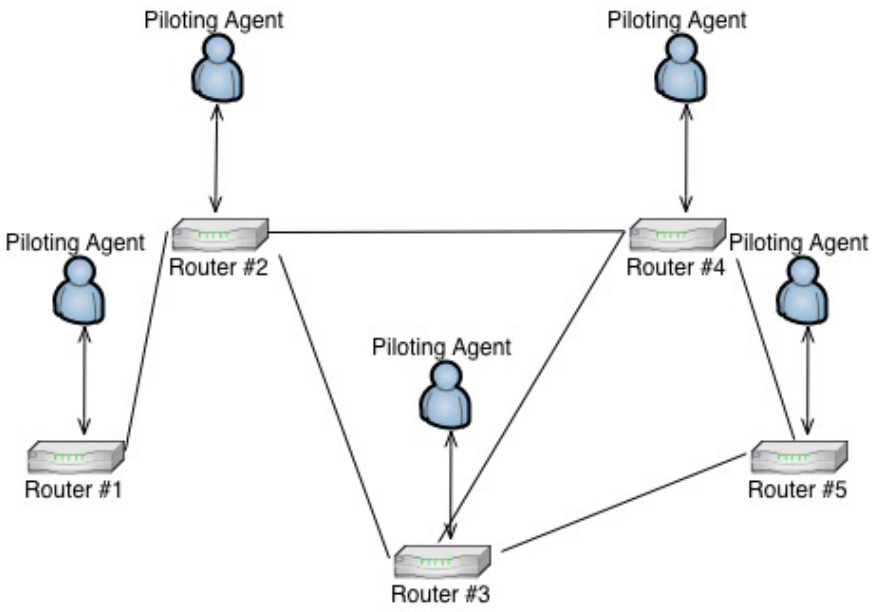


Figure 1 – Neighborhood

The piloting system operates mainly in the core network, i.e. the routers, as Figure 1 shows. Initially, agents are assigned to each router, and immediately retrieve information from the router they belong, like the routes they attend. After this step, the agent will be aware of the norms (i. e., SLA and QoS requirements) that need to couple and the normative regulation system can prohibit access to the network to those agents that violate the norms. Essentially, agents can play the following conducts: (Abiding) always abides by the norm; (Violating) may violate the norm; (Friendly) always consents to interact with others agents; and (Hiding) will avoid interact with those that violate norms.

To complete the information relevant to the piloting system, agents make contact with the others agents on their neighborhood using the behaviors defined in Section 4.1. The neighborhood in the piloting system is defined as the node (router) connected by a link, or just one hop. For example, in Figure 1, the Router #3 has as neighbors the Routers #2, #4 and #5. Thus, for each router in the neighborhood will be requested its routing table according to the routing table of the requester, its capacity to meet QoS, its current load and its average load. The latter information is requested randomly or just before a decision-making, i.e., after exceeding a threshold load. Being aware of the neighboring node ability will enable the agent to delegate or ask for routes in the inability to meet a particular request, or provide services in accordance with the requirements of quality requested.

Agent Behaviors

The piloting agents behaviors are:

- **Collect Behavior**: This behavior is responsible for collecting and storing data from the local and the neighborhood routers, the latter executed by the behavior Request Information Behavior.
- **Analyze Behavior**: This behavior analyzes the data collected and checks if they are in accordance with the quality policies required. It is also responsible for activating the decision behavior, described below.

- **Decision Behavior**: This behavior is composed of mechanisms and algorithms of decision making restricted to the data collected locally. It is possible to extend this approach to define different mechanisms for decision in accordance with the needs of the piloting system.
- **Response Information Behavior**: This behavior is responsible for serving the information request from the neighboring agents; routing tables, current and average load are sent. This behavior can be extended to address the need of other types of information in accordance with the piloting system.
- **Request Information Behavior**: Behavior responsible for requesting the local information necessary for analysis and decision making of the piloting system.
- **Create Router Behavior**: This behavior comes into play when after the decision process, the action to be taken is to create a new virtual router to meet the actual demand of the network. Thus, this behavior contact the network simulation engine requesting the instantiation of a new virtual router.
- **Create Piloting Agent Behavior**: Complementing the previous behavior, this behavior is instructed to ask the agent platform the instantiation of a new piloting agent to control the new virtual router.
- **Delegate Route Behavior**: This behavior is responsible for delegating the adherence to a particular flow to a virtual router in the neighborhood. It can be activated either after creating a new virtual router, as the perception of a neighbor with load available to meet current demand.
- **Inform Route Behavior**: Behavior used to communicate to the neighboring router, which is a flow generator, that the route was modified to conform with the actual quality criteria, and therefore the router need to update its routing table.

Acting on a QoS Failure or Malfunction

Essentially, this is a feature of the self-configuration and self-healing autonomic piloting system. When answering a particular request for data traffic, the router and therefore its pilot agent, will know which conditions and QoS must be satisfied. So, monitoring the router performance, like quality requirements, that can be configured at runtime by the network administrator, initializes the agent actions. Thus, once the agent detects a non-fulfillment, or the inability to meet quality concerns by the router, it considers whether they have enough information from their neighbors to be able to make a decision. If the agent finds that the information is outdated and that the problem is yet just a trend, it will request updated information from its neighbors, however, always monitoring the current performance of the router. After obtaining these data, the agent uses their algorithms and behaviors for deciding on their actions to solve the current problem in a decentralized approach.

Thus, the agent analyzes the routing tables of its neighbors verifies if it also serves the route that currently requires higher quality and it also checks if the neighbor has available load to provide. In a positive case, the agent will delegate this data stream to the neighboring router, performing a piloting action at runtime. However, if no neighbor is able to meet such demand, the agent will instantiate a new virtual router, associate a new piloting agent and then delegate the flow to the new virtual router.

After that, the agent must communicate with the other neighbor from where the flow is coming, to change the route. A simple scenario is shown in Section 6 .

In case of malfunction, the agent can replicate all the routes it serves to a new virtual router, and trigger an alarm for a human intervention to address the occurrence of the error, as this would be outside the scope of agent autonomy.

A Simulation Environment

We have implemented a simulation environment for the scenario previous described. We also have implemented the solution presented in Section 4 . See Figure 2 for a screenshot of the simulator. The aim of the simulation environment is to provide users with the functionality to experiment with agent-oriented pilot plans.

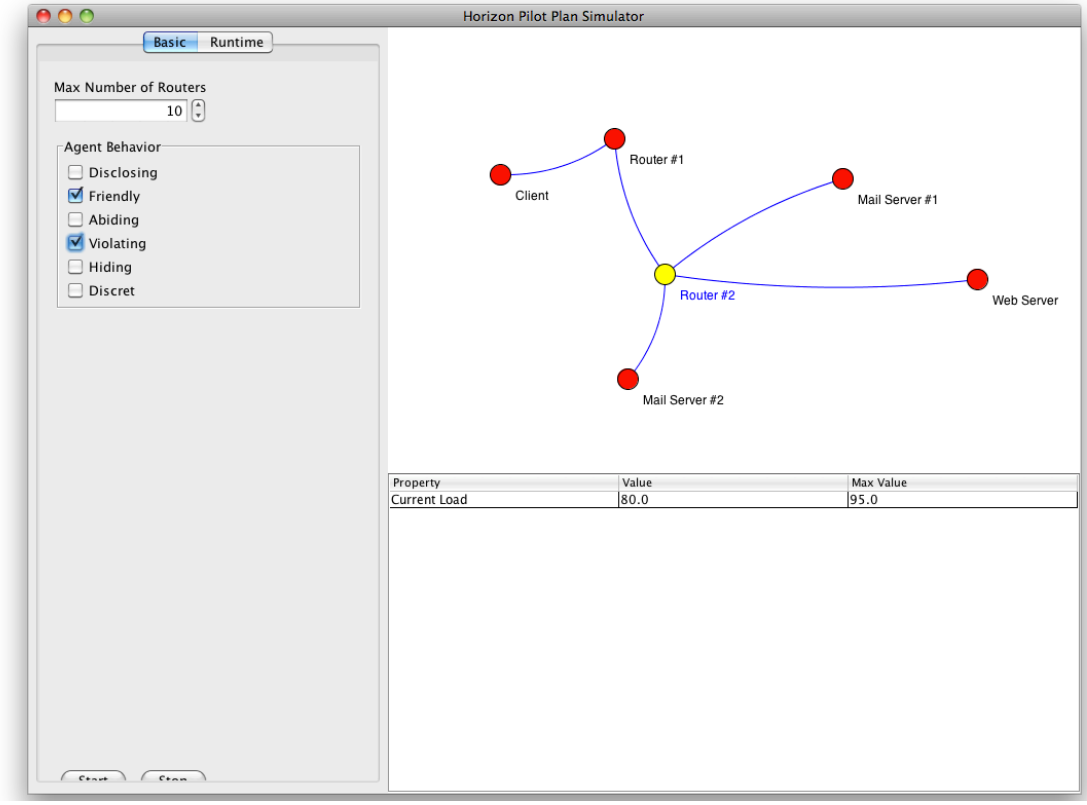


Figure 2 - Simulation Environment GUI

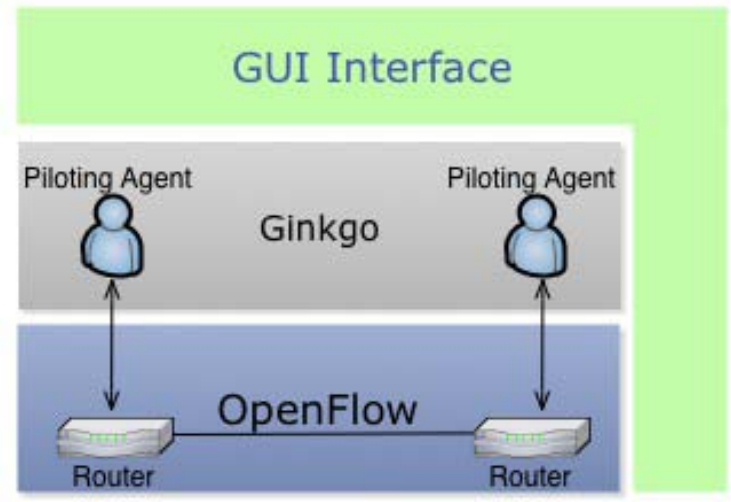
Following the architecture outlined in Section 5.1 , each node represents a virtual device. Internally the simulator maintains three different types of data:

- the virtual devices and the current value of their observed properties;
- the routes; and
- the routers that compose the virtual network.

Each time the simulation clock is incremented, the virtualization environment is consulted, and the properties values of each device are updated. Therefore, each agent is responsible for managing its local data using the API explained in Section 5.2 . Once the multiagent system decided to instantiate or change the virtual network, it executes the solution outlined in Section 4 , and wait for the new set of virtual routers/routes. The difference between the new set of routers/routes and the older one is computed

and used to instantiate a new virtual network using the API. In reality, the algorithm may behave in different ways. To illustrate these different behaviors, the simulator offers the ability to modify the different agent's behaviors (see Section 6) and simulation parameters, such as: amount of data and data transfer rate. For example, if an agent has a Violating behavior and there is a rule defining that "if the data transfer rate is below a threshold, the agent has to reject the requests to transfer data", such agent can violate such rule. Otherwise, if the agent has a Abiding behavior, it will fulfill the rule, i.e., it will not transfer data and a new virtual router must be defined.

Architecture Overview



Apêndice 1 Figure 3 - Simulator Architecture

The architecture of our proposed simulation environment is composed of three elements: (i) an user interface; (ii) a normative multiagent system; and (iii) a virtualization environment. The user interface provides users with the functionality to visualize the network and control simulation parameters. The multiagent environment implements the algorithms as presented in Section 4 . Finally, the virtualization environment provides access to virtual devices and means to instantiate and re-instantiate virtual networks.

The communication between the user interface and the simulation environment is performed via a standardized programming interface (see Section 5.2). The advantage of the programming interface is to decouple the normative multiagent system from the virtualization environment. Therefore, the simulation environment can be easily portable across multiple virtualization environments (e.g., Xen [Chisnall, 2007], OpenFlow).

GUI

The graphical user interface allows users to experiment with the proposed pilot system. The user interface provides users with the functionality to visualize the network topology, devices and routes. In addition it allows users to control some simulation parameters like the: max number of virtual routes that can be instantiated in each simulation and the behaviors that agents may assume. The GUI was implemented in Java using the JUNG library to represent the network topology. JUNG [JUNG, 2001] (Java Universal Network/Graph Framework) is a library that provides a common and

extensible language for the modeling, analysis, and visualization of data that can be represented as a network. The JUNG architecture is designed to support a variety of representations of entities and their relations, such as directed and undirected graphs, graphs with parallel edges, and so on. It also provides a mechanism for annotating graphs, entities, and relations with metadata. This have facilitated the creation of the functionalities that examine the relations between devices as well as the properties attached to each device and relation.

Ginkgo

The agents were implemented using the Ginkgo platform, through the construction of several behaviors. Using those behaviors the agents can exchange information among neighbors, store, analyze and decide what to do to attend the imposed QoS requirement, for example. The Ginkgo Distributed Network is an agent platform based on autonomic networks. It has the building blocks for the development of a piloting system for computer networks. The framework allows the creation of lightweight and portable agents, which facilitates its implementation in heterogeneous environments: routers, switches, hosts, wired and wireless networks. The agents play the role of the autonomic manager of autonomic computing. With distributed managers near its managed elements, monitoring can be done locally. The platform also allows the formation of clusters of agents in neighborhoods. Neighbors exchange information and get a situated view of the network. Thus, besides the local environment, the agent is aware of other network places. This information is stored in the knowledge base that has an information model to facilitate communication between agents. Other data repository is the policy file, which contains rules of the application. In our pilot system, rules are interpreted as norms. The behaviors described in Section 4.1 are realized as Gingko behaviors. They feed the knowledge base, perceive and predict threatening events and perform changes on the managed virtual devices. In Gingko agents also may have a dynamic planner that, with information in the knowledge base and the rules in the policyfile, changes p arameters of the behaviors and controls the life cycle of the agent. This makes possible to develop the properties of self-configuration, self-healing, self-optimizing and self-protection in the network, which promote the self-management.

OpenFlow

For the virtual network, the OpenFlow system was used. The OpenFlow provides an open protocol to program theflow table in di fferent switches and routers. A network administrator can partition traffic into production and researchflows. Researchers can control their own flows – by choosing the routes their packets follow and the processing they receive. In this way, researchers can try new routing protocols, security models, addressing schemes, and even alternatives to IP. On the same network, the production traffic is isolated and processed in the same way as today.

In order to connect the OpenFlow system and the Ginkgo platform we used the Beacon controller, which is a Java-based OpenFlow controller, built on an OSGI [OSGI, 2011] framework, allowing OpenFlow applications built on the platform to be started, stopped, refreshed, installed at run-time, without disconnecting switches. Beacon has the following features that helped us on the development of the simulation environment:

- Cross-platform - Runs anywhere Java runs (including embedded devices, e. g., switches and routers);
- Dynamic - Code and resource bundles can be started, stopped, refreshed, installed at runtime, including dependent bundles, without disconnecting switches;
- Embedded J2EE Webserver [Perrone, 2003]- Jetty [Jetty, 2011] is optionally embedded enabling a fully capable enterprise webserver;
- Unit testing - Support for JUnit [JUnit, 2011] unit testing;
- Maven [Maven, 2011] - Beacon can be built using Maven, and exported to Maven and P2 repositories.
- Performance - Beacon has been tested and shown to service 250,000 L2 switch Packet-In requests per second in single threaded mode on a 2.4ghz Core 2 processor using 512MB of RAM. Widening the thread count to 3 increases performance to 340,000 Packet-Ins/s.

Programming Interface

The programming interface defines how to integrate the simulation environment with any virtualization environment. This section provides a comprehensive description of the proposed programming interface. The aim of the interface is to allow the simulation environment to perform tasks such as: (i) obtaining the available physical devices and routes; (ii) getting the instantiated virtual network; (iii) observing values of devices properties; and (iv) reconfigure the virtual network. Figure 4 provides an overview of the concepts that compose the interface.

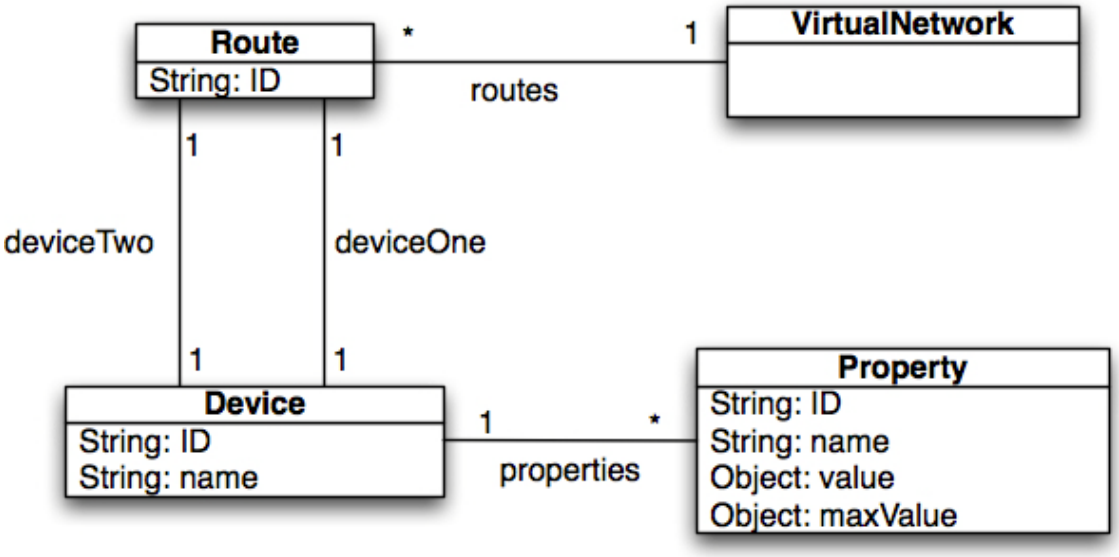


Figure 4 - API Class Diagram

The Device concept represents the physical network devices, such as routers. Each device has a unique ID, a name and a set of Property elements. A property is a pair of name and value uniquely identified by an ID. The routing tables are represented by the Route concept. A route maintains the linked devices and is identified by an ID. A virtual network is a sub set of the set of routes that compose the physical network.

In order to manage the virtual network, the programming interface provides a function that supports the piloting system to manage the virtual network by adding and removing routes and devices, accordingly. Therefore the setVirtualNetwork (addRoutes, removeRoutes) function receive as input two parameters: (i) the addRoutes parameter is a list of routes that will be part of the virtual network; and (ii) the removeRoutes parameter is a list of routes that will be removed from the virtual network. The following are the six functions that can be used to obtain information from the network:

- devices = getDevices() returns all devices that are part of the virtual network as descriptors used to refer to the devices in subsequent calls.
- device = getDevice(deviceID) returns the device descriptor identified by the deviceID. When the device does not exist it returns an invalid descriptor. The deviceID is only a symbolic link and must be managed by the virtual environment plugin.
- value = getPropertyValues(deviceID, propertyID) returns the value of the property identified by the propertyID from the device identified by the deviceID. When the property and/or device do not exist it returns an invalid descriptor.
- routes = getRoutes() returns the routing table of all devices as route descriptors. The routes are used to compute new virtual networks.
- routes = getRoute(deviceID) return the routing table of a given device identified by the deviceID.
- routes = getVirtualNetwork() return the set of routes that defines a virtual network.

Usage Scenario

In order to demonstrate the piloting system with the driving simulation environment, we implemented a virtual network in OpenFlow and their respective piloting agents in Ginkgo. For the sake of simplicity, we used a linear topology to demonstrate the application of the simulator and the proposed piloting plane. In Figure 5 we have the initial representation of the topology used. After the instantiation of the network elements, the agents begin the process of data collection. For this scenario, we have as a neighbor of router #1 the router #2 and vice versa. Therefore, the router #1 will run the Collect Behavior for collect its routing and cargo information, as well as the Request Information Behavior to request relevant data from the router #2, which responds through the Response Information Behavior. The router #2 performs the same collect process.

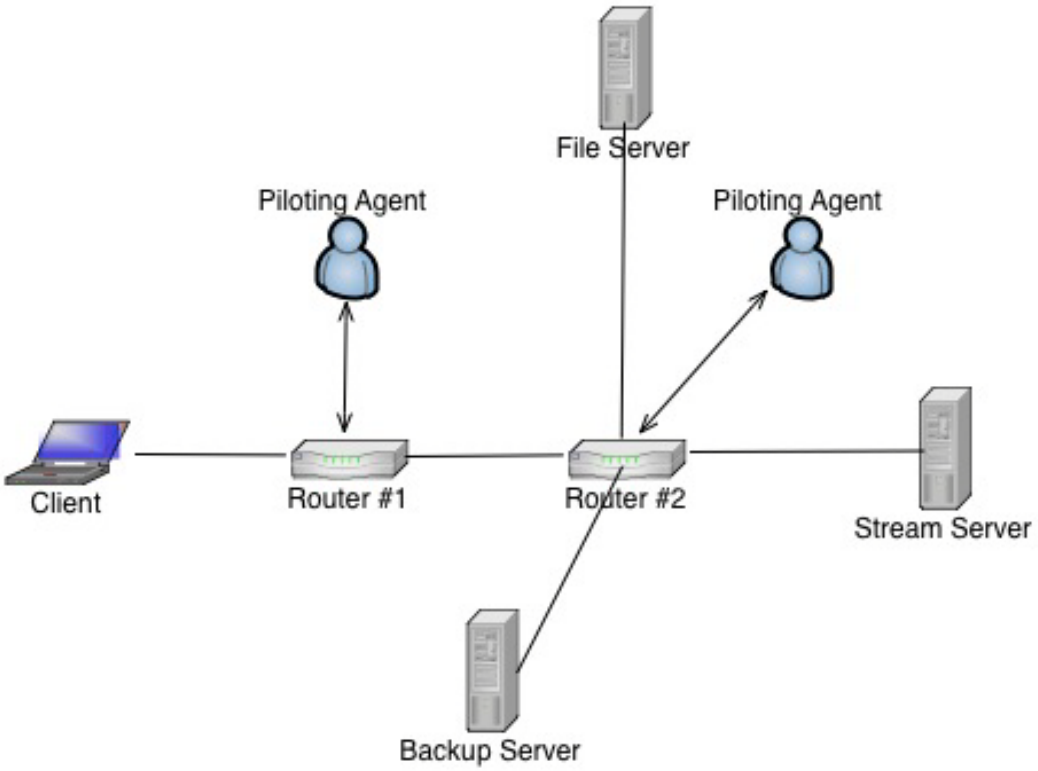


Figure 5 - Virtual Network Topology

To begin the simulation process, the Client requests a stream service from the Stream Server, which is attended by a route composed by two virtual routers in the network. This stream request must meet certain criteria of quality of service (QoS), agreed in the Service Level Agreement (SLA). Initially, the routers are idle, and promptly attend to the requested stream with the required quality. We can observe in Figure 6 that the router #2 is in accordance with the QoS criteria.

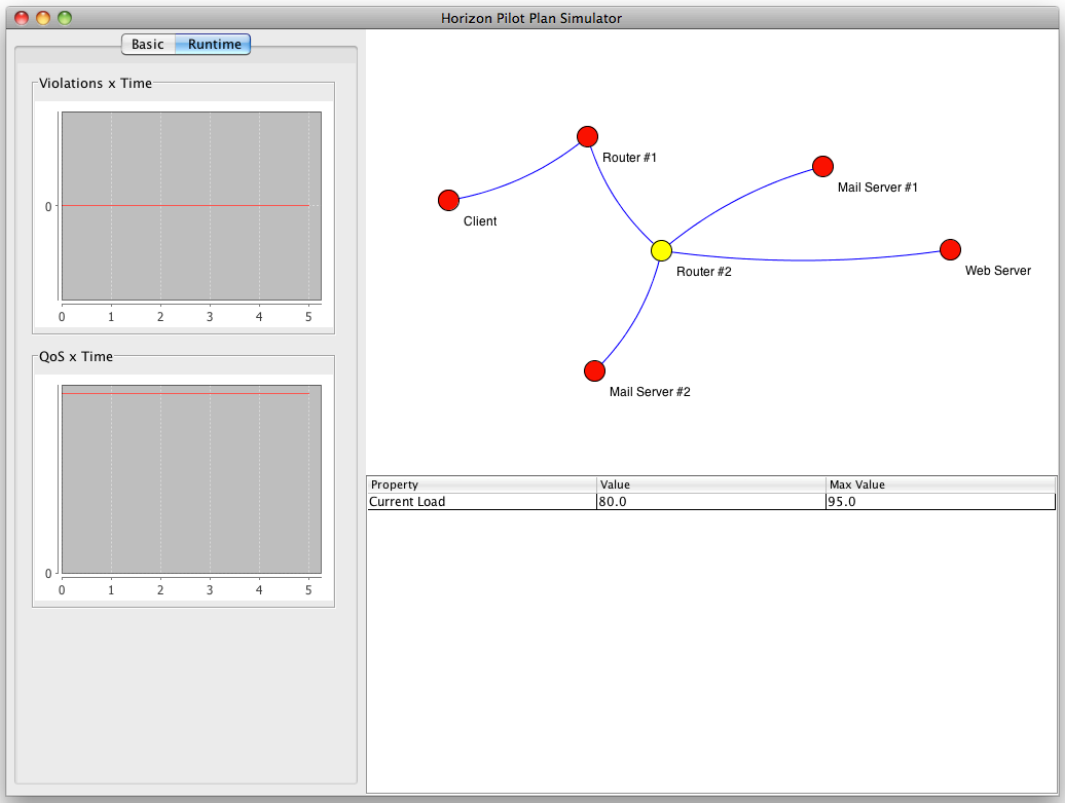


Figure 6 - Router #2 respecting the QoS criteria

When starting transmission, the piloting agents verify the service quality attendance. However, to generate a disturbance in this scenario, we started a massive data transfer from the File Server to the Backup Server. Notably, this transfer will compromise the router # 2, which becomes overloaded and unable to meet the quality required by the stream service, which initiates a process of quality norm violation. Figure 7 shows the router #2 overloaded and so violating the QoS norm.

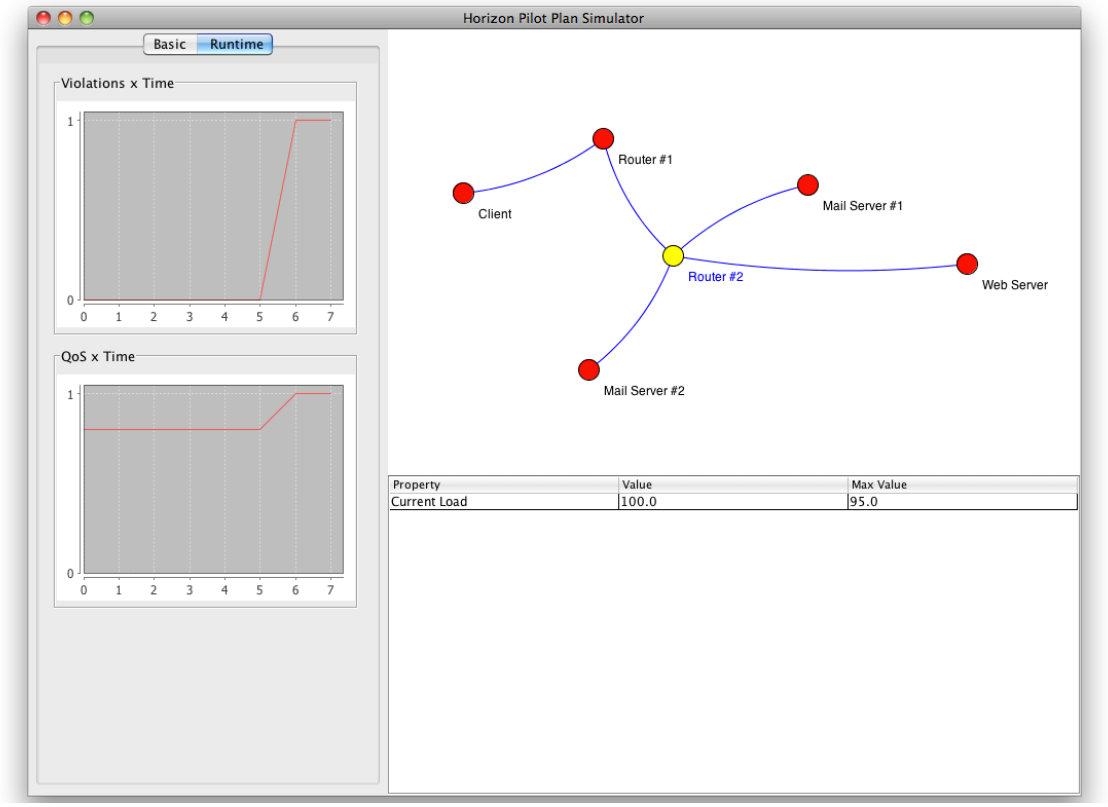


Figure 7 - Router #2 violating the QoS norm

Thus, the piloting agent realizes this disorder, and as a way to solve it starts searching for neighbors who have load available and meet the same segment of the route. However, as can be seen, no neighbor is able to meet this request. Thus, the piloting agent of the router #2, communicates with the Beacon controller, using the Create Router Behavior, to request the instantiation of a new virtual router on the network. After the construction of this new router #3, the piloting agent runs the Create Agent Behavior to allocate an agent to the router #3. Then the route in question, from router #1 to the Stream Server, is delegated to the router #3 through Route Delegate Behavior, and finally the Inform Route Behavior informs the router #1 about such modification in the route, as we can see in Figure 8.

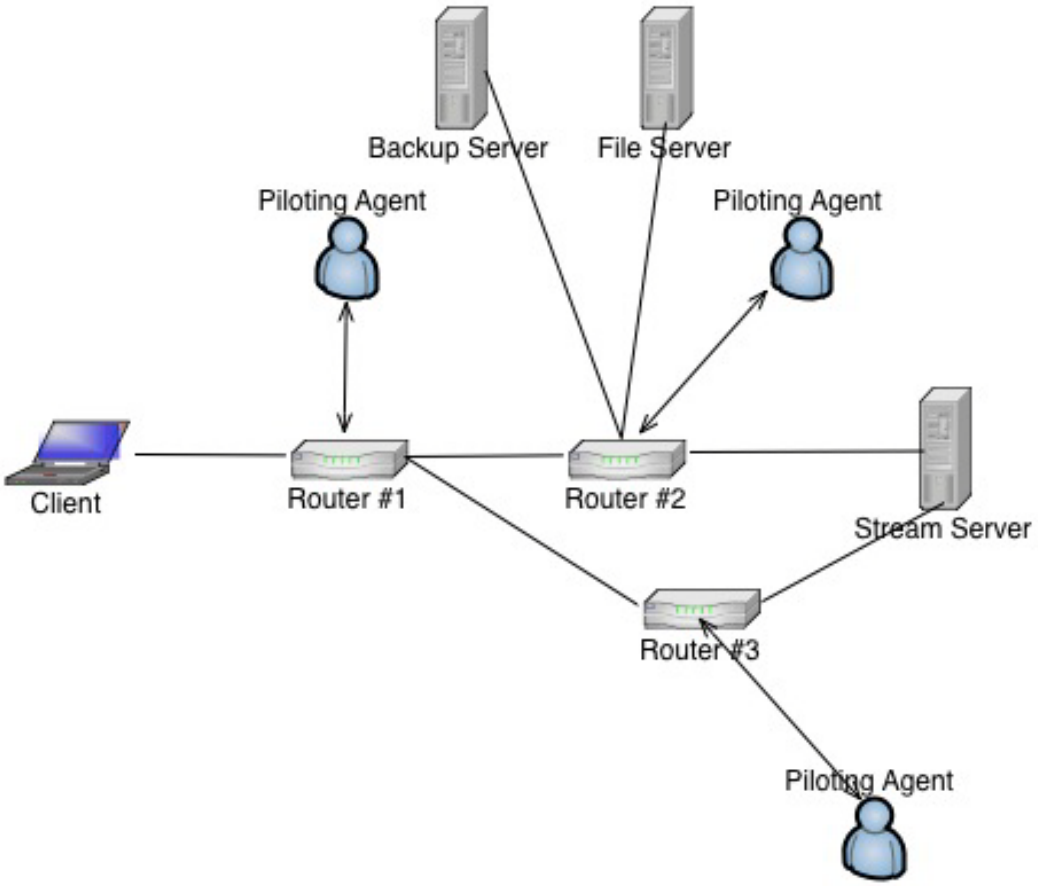


Figure 8 - Final Topology

Through the Figure 9, we can view the information being displayed on the interface of the simulation environment. It is important to note that between the period of non-compliance of quality criteria and its solution, the piloting agent of router #2 is violating the norms of quality. Also, in the same figure we also note the fall of the QoS attendance of the transmission, and their improvement after the creation of router #3.

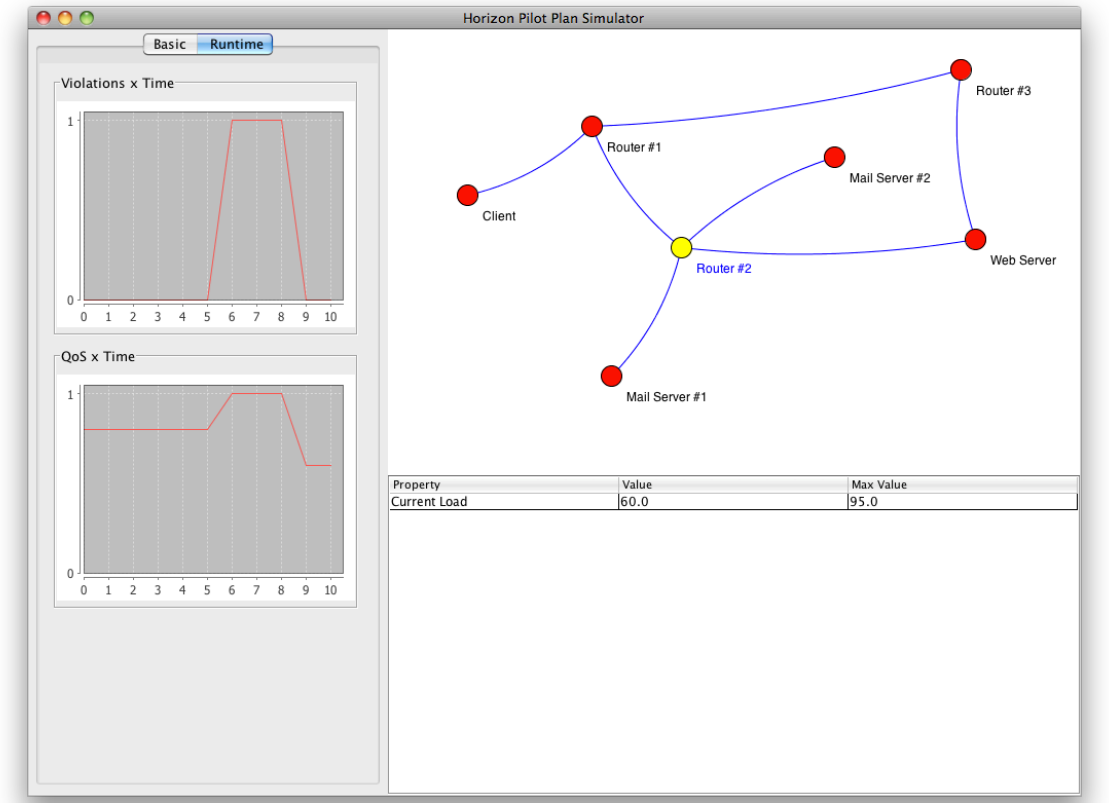


Figure 9 - Virtual Network fulfilling the QoS norm

Conclusion

This work presented a conceptual piloting system based on self-organizing and normative multiagent system, which taking advantage of the intelligent decisions performed by piloting agents, needed to govern and adapt the virtual network in response to changing context. Besides the piloting system, we have implemented a simulation environment which supports users to test and analyze different normative and organizational configurations of the piloting multiagent net-work.

Now, we are working on the evolution of the simulation environment in order to enable the piloting agents to execute new behaviors and adopt new normative conducts. In addition, we intend to enable users to apply different self-organizing strategies and verify the system behavior in face of the application of such strategies.

References

TURNER, J.; TAYLOR, D. Diversifying the internet. In: Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE, vol. 2, pp. 6 pp. –760, December 2005.

FEAMSTER, N.; GAO, L.; REXFORD, J. How to lease the internet in your spare time. In: SIGCOMM Comput. Commun. Rev., vol. 37, pp. 61–64, January 2007.

ANDERSON, T.; PETERSON, L.; SHENKER, S.; TURNER, J. Overcoming the internet impasse through virtualization. In: Computer, vol. 38, no. 4, pp. 34 – 41, April 2005.

GINKGO NETWORKS. **Ginkgo distributed network piloting system**. Technical Report. Ginkgo Networks, Sept. 2008.

McKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; L. PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. OpenFlow: Enabling innovation in campus networks. In: ACM SIGCOMM Computer Communication Review, vol. 38, no. 2, pp. 69–74, Apr. 2008.

FERBER, J. **Multi- Agent System: An Introduction to Distributed Artificial Intelligence**. Harlow: Addison Wesley Longman, 1999.

BROOKS, R. A. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, Vol. 2, No. 1, March 1986, pp. 14–23; September 1985.

BERGER, M.; ROSENSCHEIN, J. S. When to Apply the Fifth Commandment: The Effects of Parenting on Genetic and Learning Agents. AAMAS'2004, New York, USA. ACM, pp 19- 23, July 2004.

DUTTA, P. S.; JENNINGS, N.; MOREAU, L. Cooperative Information Sharing to Improve Distributed Learning in Multi- Agent Systems. Journal of Artificial Intelligence Research, Vol. 24, pp 407- 463, 2005.

TUOMELA, R. **The Importance of Us: A Philosophical Study of Basic Social Norms**. Stanford University Press, 1995.

TUOMELA, R.; BONNEVIER-TOUMELA, M. **Social norms, task, and roles**. Technical Report HL-97948, University of Helsinki, Helsinki, 1992.

TUOMELA, R.; BONNEVIER-TOUMELA, M. Norms and agreements. European Journal of Law, Philosophy and Computer Sience, 5:41–46, 1995.

SERUGENDO, G. Di M.; FOUKIA, N.; HASSAS, S.; KARAGEORGOS, A.; MOSTFAOUI, S. K.; RANA, O. F.; ULIERU, M.; VALCKENAERS, R.; AART, C. Van. Self-organisation: Paradigms and applications. In: Giovanna Di Marzo Serugendo, Anthony Karageorgos, Omer F. Rana, and Franco Zambonelli, editors, Engineering Self-Organising Systems: Nature-Inspired Approaches to Software Engineering, volume 2977 of LNCS (LNAI), pages 1-19. Springer, May 2004.

SYCARA, K. Multiagent Systems. Artificial Intelligence, vol. 10, no. 2, pages 79-93, 1998.

SERUGENDO, G. Di M.; GLEIZES, M. P.; KARAGEORGOS, A. Self-Organisation in MAS. Knowledge Engineering Review 20(2):165-189, Cambridge University Press, 2005.

CHISNALL, D. **The Definitive Guide to the Xen Hypervisor**. Prentice Hall, 2007.

JUNG. Available at: http://jung.sourceforge.net/ Accessed in July/2011.

OSGI. Available at: http://www.osgi.org/Main/HomePage Accessed in July/2011.

PERRONE, P. J. J2EE Developer's Handbook. Indianapolis, Indiana: Sam's Publishing, 2003.

Jetty. Available at: http://jetty.codehaus.org/jetty/ Accessed in July/2011.

JUnit. Available at: http://www.junit.org/ Accessed in July/2011.

Maven. Available at: http://maven.apache.org/ Accessed In July/2011.

SHERIDAN-SMITH, N.; O'NEILL, T.; LEANEY, J.; HUNTER, M. A policy-based service definition language for service management. In: Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP, April 2006, pp. 282–293.

DERBEL, H.; AGOULMINE, N.; SALAN, M. Anema: Autonomic network management architecture to support self-configuration and self-optimization in ip networks. Computer Networks, vol. 53, no. 3, pp. 418 – 430, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/B6VRG-4TW14YJ-3/2/41147806c839b89928697fc9b724d880

JENNINGS, B.; MEER, S. van der; BALASUBRAMANIAM, S.; BOTVICH, D.; FOGHLU, M.; DONNELLY, W.; STRASSNER, J. Towards autonomic management of communications networks. Communications Magazine, IEEE, vol. 45, no. 10, pp. 112–121, October 2007.

KEPHART, J. O.; CHESS, D. M.; The Vision of Autonomic Computing. Computer, vol. 36, no. 1, pp. 41-50, Jan. 2003