Towards autonomic networking and self-configurating routers

The integration of autonomic agents

Thomas Bullot[1,2] and Dominique Gaiti[1]
(thomas.bullot@utt.fr) (dominique.gaiti@utt.fr)

(1) - ICD - FRE CNRS 2848, Université de Technologie de Troyes,
BP2060, 10010 Troyes cedex, France.
(2) - GINKGO-Networks, 8 rue du capitaine Scott 75015 Paris

Abstract. IP Networks, and particularly the Internet, were proposed to be a simple and robust support for heterogeneous communications. This implies that only basic controls have to be done by network elements. Connection management, along with transport, and more generally communication management, has to be done by the terminals. For example, error detection mechanisms, error recovery mechanisms with "Slow Start", are implemented within the transport protocol, managed by the terminals. However, integration of new services and increasing need for QoS require the network to be more and more flexible and adaptive. New algorithms and protocols are then proposed to address these issues, and include new configuration layers. Manual configuration of such network architectures is then very complex, if not impossible. We think that future core network elements will have to be more adaptive, but also more autonomic. Auto-configuration is indeed a necessary condition to integrate new services in the network. We believe that auto-configuration requires new knowledge provisioning and computing policies. This paper then presents an architecture of software agents, collaborative and autonomic. These agents are embedded inside the routers. Their role is to share local and situated knowledge, in order to control and optimize the existing control mechanism of the router.

1 Introduction

Since several years, evolution of networks includes the integration of new services. This achievement implies the setting of different features and tools. On one hand,

new control mechanisms (traffic engineering, QoS, security, etc…) are then proposed. On another hand, a new perspective of network evolution, more and more present in the discussions, is autonomy. It refers on creating a network that has self-configuring, self-healing, self-optimizing, and self-protecting features [IBM03], in order to adapt to new situations in the context.

To achieve these issues, the new mechanisms need a huge quantity of numeric and/or symbolic knowledge. Provisioning, computing and representation of this knowledge are for now achieved separately at a very low level by each existing control mechanism. We think that high level information cannot be provisioned at a larger scale by existing mechanisms, because this knowledge is complex, and it has to be more specifically provisioned.

This paper presents an architecture of software agents, collaborative and autonomic [FER99]. These agents are embedded inside the routers. Their role is to share local and situated knowledge, in order to control and optimize the existing control mechanism of the router. Our work is based and inspired by several papers in the literature, including [MES03] which proposes a multi-agent based architecture to share knowledge and control DiffServ; [MER03] which presents a 2 layer end-to-end adaptive monitoring mechanism based on collaborative agents. In a similar field, [BIE98] proposes to base the network management on a mobile agent system.

We first describe in section 2 the internal basis structure of an agent. In the third section are described the interfaces of this agent with its environment. In the end, last section proposes an application of this technology to control and optimize the routing algorithm OSPF, using high-level, local, and situated elements of knowledge. The steps of the conception of an agent are not detailed in this paper.

2 Basis structure of an agent

An agent is made of a decision core and several modules (fig. 1). Decision core is the kernel of the application. It is based on synchronous execution of several “behavior units”, and is able to compute received data and observed data, in order to decide of an action.

Modules are the sensors and effectors of the agent. These are interfaces with the environment of the agent, which allows the behavior units to observe a given part of the functional environment (reading an element in a MIB, receiving inter-agent messages, etc…), and act on it (configuring equipments, send inter-agent messages, reporting events to the network operator, etc…).

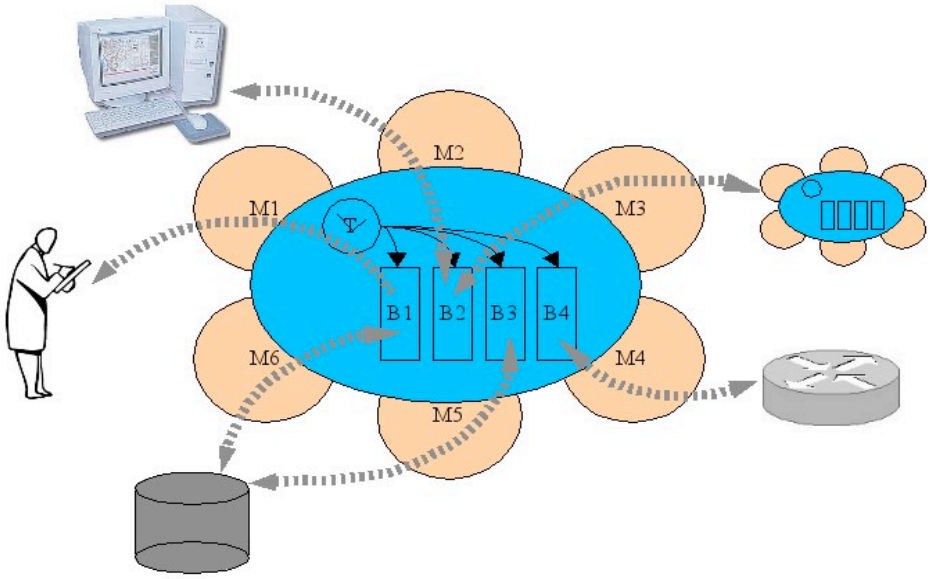

Fig. 1. The agent within its environment

In this section the decision core and the modules mechanism are described in details.

2.1 Decision core

The decision core is based on one or more behavior units. These units are execution units, synchronized by an internal timer. They are executed in a given order, at a fixed rate.

There are three different kinds of behavior units:

- Observation and listening units, which are able to gather information about network elements and neighbor agents ;
- Computing units, which extract useful information from the data gathered by observation units, and compute a "good" decision for the network element configuration ;
- Action units, which apply actions decided by the computing units.

These are executed sequentially, in order to plan mechanisms over several behavior units. Decisions are taken after information was collected. Each one of these behavior units can access the modules they want. Actions decided by these units can use primitives of available modules ("send a message", "check a mailbox", "set a parameter", etc…). Each of these can access the controlled equipment, send messages to the neighbors, and update a remote database.

2.2 Modules

Modules are interfaces between an agent and its environment. Primitives are described and behavior units can access them. Development of these modules is strictly separated from the development of the application. It depends on environment (architecture, hardware, technical means, etc…) and on primitives to provide.

For example, the primitive « send a message » of a communication module can either: open a TCP connection with a neighbor to send an ACL message; send a simple string in UDP, without opening any connection; send an only Ethernet frame including the whole desired information; invocate a Java-RMI method on a remote agent; etc… There is a huge amount of possibilities. On the agent side, only the functionality is seen.

2.3 Needed support features

When building the different agent behaviors, one will need some basic functionalities (change a configuration parameter on the controlled equipment, send a message to a peer agent, etc…). These functionalities have to be provided to agents by the primitives of available modules. These modules, then, use some primitives in the controlled equipments (where the agent is embedded). If the agent, for example, has to be able to communicate with peers, it will need a communication module, which in turn will be based on available primitives in the equipment (a router with a UDP transport layer, for example).

3 Agent interfaces

These interfaces are classified in 3 categories (fig. 2): Agent-router interfaces, agent-agent interfaces, agent-human interfaces.

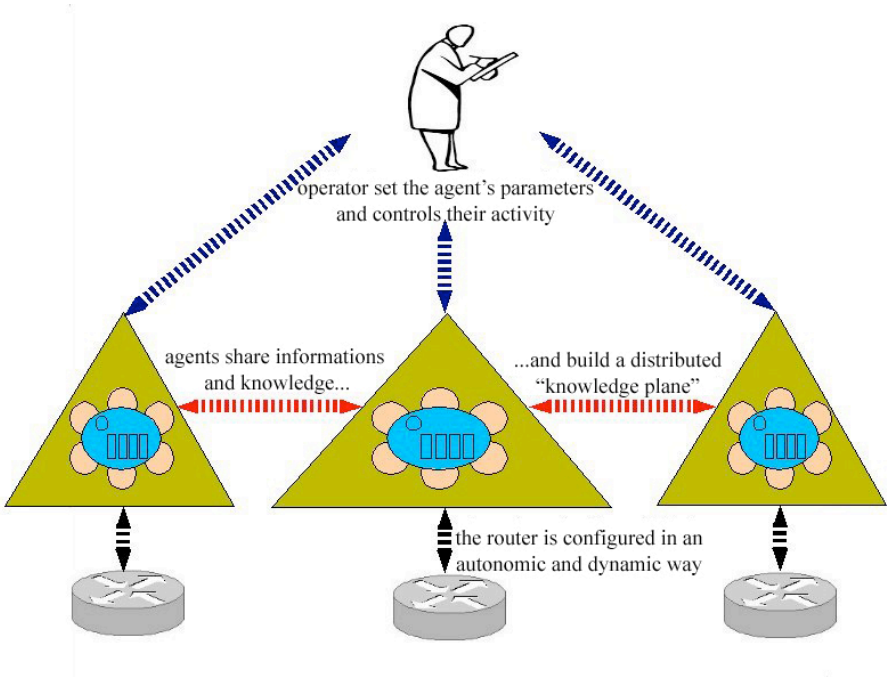

Fig. 2. Interfaces between the agent and its environment

3.1 Agent-to-router interface

The agent-to-router interface is the most important interface: it allows the agent to control routing and QoS functions in the router.

This could be a local interface, if agents are embedded within routers (fig. 4); it could also be a remote interface, if agents cannot be embedded within routers (fig. 3). In any case, it can be based on IP-level protocols (telnet connection, SNMP, etc…).

Accessing the router parameters can be done in command lines via a telnet connection, or by consulting MIB databases. Generally, gathering data can be done within 2 modes: polling (active gathering of information from the router) or trap reception (alarm messages sent by the router when a threshold is reached).

Accessing the configuration always depends on the application. It could even not be necessary, when applications do not need to re-configure the router (i.e. monitoring applications). If it is necessary, it can be done in command lines via the router shell, or via specific configuration protocols like netconf.

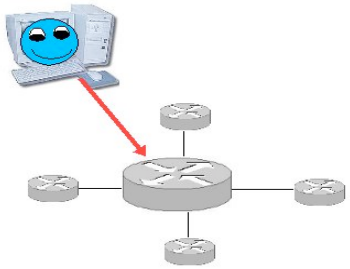

Fig. 3. Deploying the agent : remote control agent

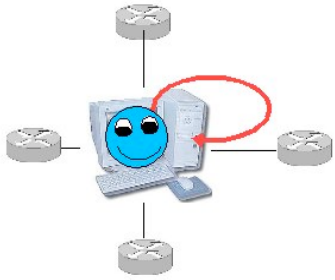

Fig. 4. Deploying the agent : local control agent

3.2 Agent-to-agent interface

Agent-to-agent communication is a key issue, and one of the most important innovations provided by multi-agent systems. Transversal sharing of information by agents allows one to build a global distributed "knowledge plane". Each agent can build its own situated view of the network state. Addition of these different situated view results in what one can call a distributed knowledge plane. The situatedness of agent knowledge is a key point in our architecture: it ensures that our system can be scalable.

This transversal, peer-to-peer, and not hierarchical communication model is an answer to one of the most important issues in the domain of networks: how to build and maintain a global network state representation, without being stopped by scalability problems? Building in real time a unique and centralized view of the network is only possible in very small networks. On the contrary, if one accepts that this representation is not centralized, not always complete and deterministic; our architecture is a suitable answer.

In order to exchange data in a coherent way, a suitable representation of these data has to be adopted. On the suitability of this model will depend the efficiency of the application. It is necessary for example to find an adequate level of granularity.

Exchange protocols can also be a key issue of an application. Depending on the kind of protocol used, one can sometimes with very few messages exchange a huge amount of information. These protocols also have to be extremely robust: each agent has to be autonomic; it is a key quality of our system! The stability of an agent can never rely on the reception of a message. Every situation must be faced and managed. One should avoid explicit requests. Instead, one way information messages are preferred.

3.3 Agent-to-human interface

Technically, the agent-to-human interface is not a crucial interface. However, it's necessary for a human operator to observe and influence the evolution of the system. Monitoring facilities can be very precise, in order to allow a human operator to understand and validate or invalidate some of the agent-decisions in real time. On the contrary, it can be a very synthetic high-level monitoring, to observe the evolutions of the agents.

Agent configuration, in the same way, can either be very synthetic at a high level or more technical at a lower level. It should generally not be too precise, because if the agents are so difficult to configure as the network itself, utility of these agents is much reduced. Ideally, configuration of the agents should be down automatically by the agents themselves. To achieve this kind of auto-configuration skills, the agent has to be built to need as less information as possible. Its efficiency will always rely on the amount of information it's got to build a representation of the network, but it has to be working with very few information.

4 Automatic configuration of OSPF protocol using such agents

An example of agents applying on network functionality is auto-configuration of OSPF protocol.

OSPF is a dynamic routing protocol. As a routing protocol, it is a task achieved by all of the routers in the network. As it is dynamic, it should be done depending on the context and the local environment of nodes. Because of these 2 reasons, this example is a good application for our agent architecture.

The work of the agent consists in observing the current situation, and applying OSPF parameters reconfigurations, in real time.

4.1 Principles

OSPF is a dynamic routing protocol, which computes in real time all the routing tables depending on the weight affected to each interface. One route will be chosen to send packets from one point to another if the sum of the weights of each used interface in this route is minimal in the network.

Interface weights are then a key issue for choosing routes in OSPF. Choosing these weights is a complex issue. Many works refer to the general routing problem. [BER92] proposes a set of solutions to this problem, and [WAN99] studies different versions of this problem. The particular OSPF weight setting problem, on the contrary, has been proven to be NP-Hard [FOR00], and many heuristics has been proposed to address it. A generally accepted heuristic to choose these weights was proposed by Cisco [CIS98]: the weight of an interface should be proportional to the inverse of the capacity of that interface. Very capable interfaces are then more used than less capable interfaces. However, this heuristic doesn't take in account topology and current load of links, and sometimes causes misuse of small external links and overload of big main interior links.

A lot of works on the OSPF Weight Setting Problem [FOR00], and particularly using techniques based on artificial intelligence [ERI01], encountered a reasonable success [YE02], but it is often about off-line optimization techniques, based on such pieces of information that "demand matrix", which are very often unavailable.

How, then, to choose a "good" configuration of OSPF weights, at execution time, depending on topology and the current state of the system, without encountering scalability problems (i.e. with no centralized computing).

To address this issue, we propose our autonomic agent architecture. As far as we know, no work before has addressed this problem using multi-agent systems. At a fixed rate, agents observe the state of the different interfaces of their controlled router. They then broadcast a representation of that information to their neighbors. Each agent is informed of the load of its interfaces and of the state of its neighbors. It will then be able to recompute the weight of its interfaces.

4.2 Agents behavior

The implemented behavior can decide of 2 independent actions: on one hand, it can decide to change the weight of one of its interfaces; on the other hand, it has to decide what the new weight to set is. These decisions have to be separated, because changing a weight causes the whole network to be flooded with the topology with the new weights, and all the routing tables to be recomputed. It is important then to decide very rarely to change the weights. This decision can rely of different pieces of information as the new weight to set.

The choice of interface weights is based on the current load of these interfaces, and on the state of the routers directly linked to these. It's indeed very important, before deciding to redirect more traffic on an interface, to see if the adjacent router to this interface is already loaded or not.

In the following example (fig. 5), router-agent O has to choose OSPF weights of its interfaces. Available information is showed on the figure: the load of each one of its interfaces (cursors near each interface), and the state of nodes directly linked to these interfaces (color of the nodes). In this case, router-agent O has to set a weight to 3 interfaces, numbered from 1 to 3. For the interface 1, the current load of the interface is high, and the adjacent router (A) is in a "critical" state (red). Agent O's decision will then surely tend to increase the weight of this interface, in order to reduce the traffic passing through it. For interface 2, the load is high, but the state of the adjacent router (B) is "good" (green), so one can guess that the weight of this interface can remain the same. For interface 3, the load is low and the state of the adjacent router is "good", then the decision may be to decrease the weight associated to this interface.

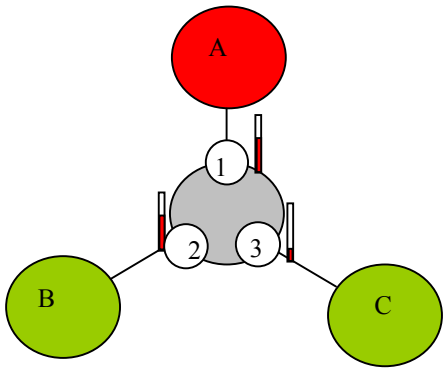

Fig. 5. An example of a weight setting situation

On the contrary, deciding to effectively change the weights of the interfaces in a router (and, then, to reset the whole OSPF protocol in the network) does not rely directly on the load of interfaces, but on the global state of the router. This state takes in account the load of interfaces (max load, average load, etc…) and an evaluation of the state of environment (state of the neighbor routers). To avoid multiple OSPF resets at the same time in the same neighborhood, each agent sends an "I reset!" message to all its neighbors (on 1, or 2, or 3 hops). Multiple resets in a same neighborhood could indeed cause oscillation and strongly decrease the performance in the network. To avoid these problems, we also propose to take into account 2 kinds of influences: stabilizing influences and destabilizing influences. A positive state of the neighbors is a stabilizing influence. In the contrary, negative state of the neighbors is a destabilizing influence. Heavy load of an interface is a destabilizing influence, when light load of an interface is a stabilizing influence.

To avoid oscillation around a not-perfect-but-optimal configuration, we included within the agent behavior a habituation phenomenon. That means that a data has no sense by its own. Each data taken into account by the agent is pre-computed to compare it with previous identical data, to know if it is "particularly high", "normally high", "surprisingly high", etc… More precisely, each data is compared with its minimum, its maximum, and its average. This pre-computing is done by normalization (fig.6). Our normalization is done with the following system:

$Xnorm = [(X - Xmin)/Xavg]*0.5$ if $0 \leq X < Xavg$

$Xnorm = 0.5 + [(X - Xavg)/Xmax]*0.5$ if $Xavg \leq X < Xmax$

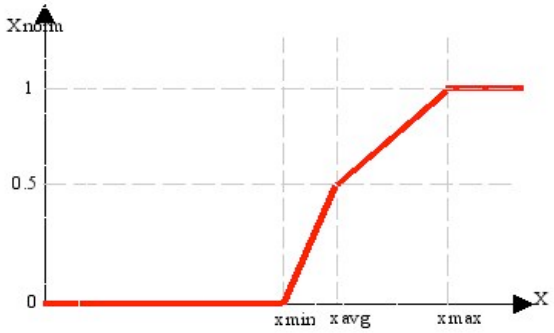

Fig. 6. A representation of our normalizing operation

5 The simulation conditions

5.1 Motivations

The main motivation of this simulation is to show how the system works, and what its objectives are. This is definitely not a performance test, and one should take into account mechanisms, and not values. Values are given as a piece of illustration information. For the same reasons, the simulated traffic is uniform.

5.2 Data

The network topology used in our simulation (fig. 7) represents a network that is similar to the one of the British operator British Telecom. This topology has been chosen because it represents a network that is technically credible, highly meshed, with enough nodes. The nodes are distant enough (distance up to 5 hops between 2 nodes).

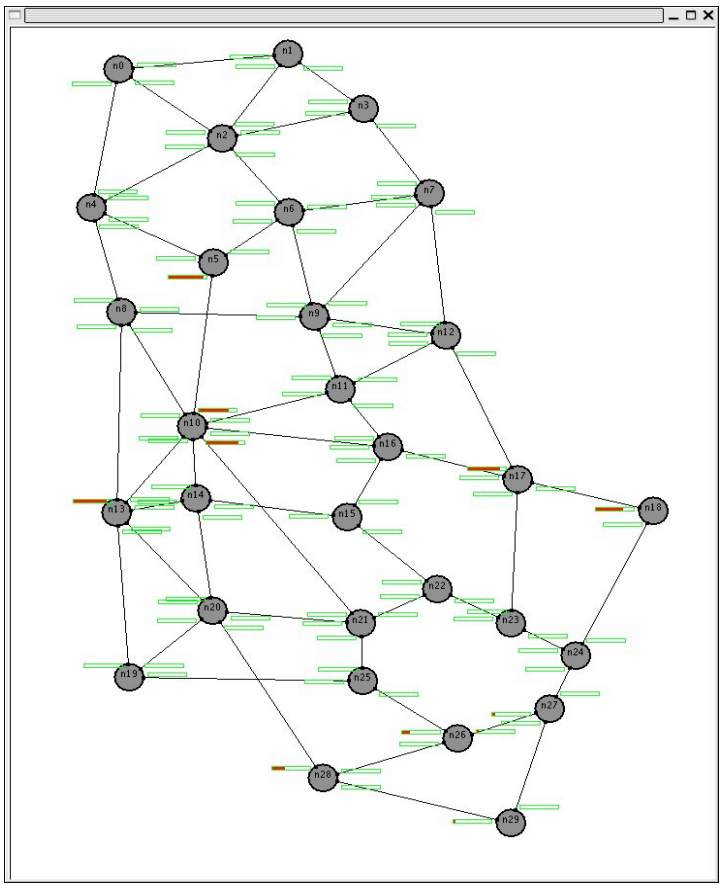

Fig. 7. Topology used in this simulation

The traffic introduced is not even a nearly realistic traffic model. Every node in the model sends a uniform traffic stream to each other node in the network. The only aim is to overload the network, in order to push the agent system to try and find an optimal solution.

5.3 Simulation results

Given results are to study qualitatively. Our main evaluation criteria in a given instant is the number of overloaded interfaces at this instant. We just consider that, approximatively, the global load of the network is linked with the number of overloaded interfaces. This hypothesis is not exactly true, because an overloaded interface can reject a huge number of packets, when two overloaded interfaces can be just a little bit overloaded, and reject a very few packets… However, this is not the most common situation, and even if our results "only" show a very remarkable tendency of reducing the number of overloaded interfaces, one can think that we found a way to compute "good" weights in OSPF.

A first test allows us to observe the OSPF behavior without any agents (fig.8), configured following the Cisco heuristic: weight(interface) = 1/capacity(interface).

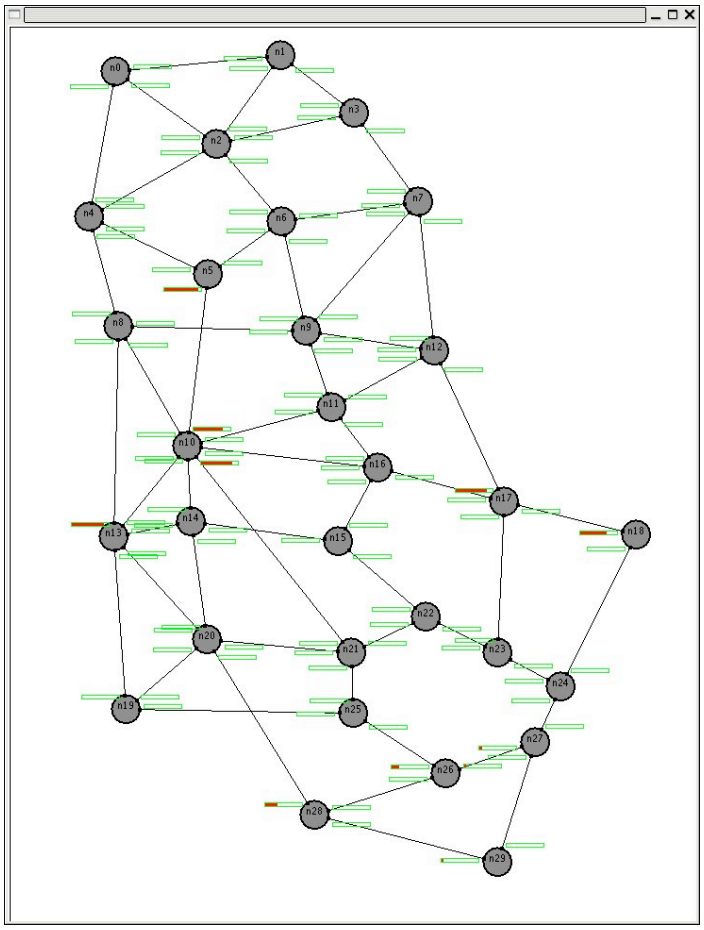

Fig. 8. OSPF behavior without any agent

One can see that 6 interfaces are overloaded. One in node n5, two in node n10, one in node n13, one in node n17, and one in node n18. These interfaces are clearly overloaded, and the other interfaces are not.

When activating agents, one can observe a few transitive phases (fig.9). Duration of these phases is not studied here. Our aim in further studies will be to obtain duration of a transitive state that is proportional to the "stability" of the configuration (cf. stabilizing and destabilizing influences, in previous part). One data to study will also be the number of these transitive states. It has been told in previous parts that the cost of resetting OSPF is very high. It is capital to reduce the number of these resets at its minimum, and to converge as quickly as possible to a stable state.

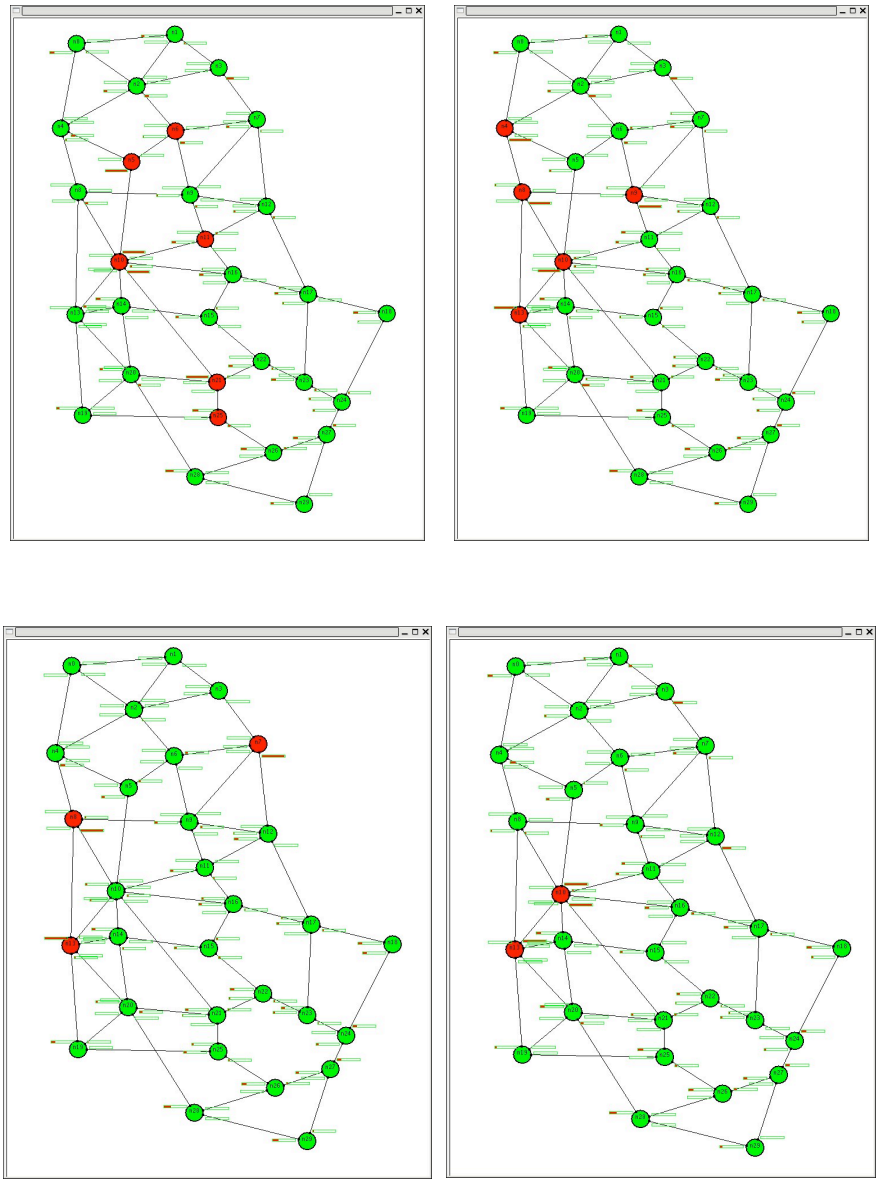

Fig. 9. Succession of transitive states

Then, one can observe a succession of stable states. Stability of these states relies on the state of all agents in the network. Agents are influencing each other, and push to a stabilization of the global system. An example of a stable state is represented in fig. 10.

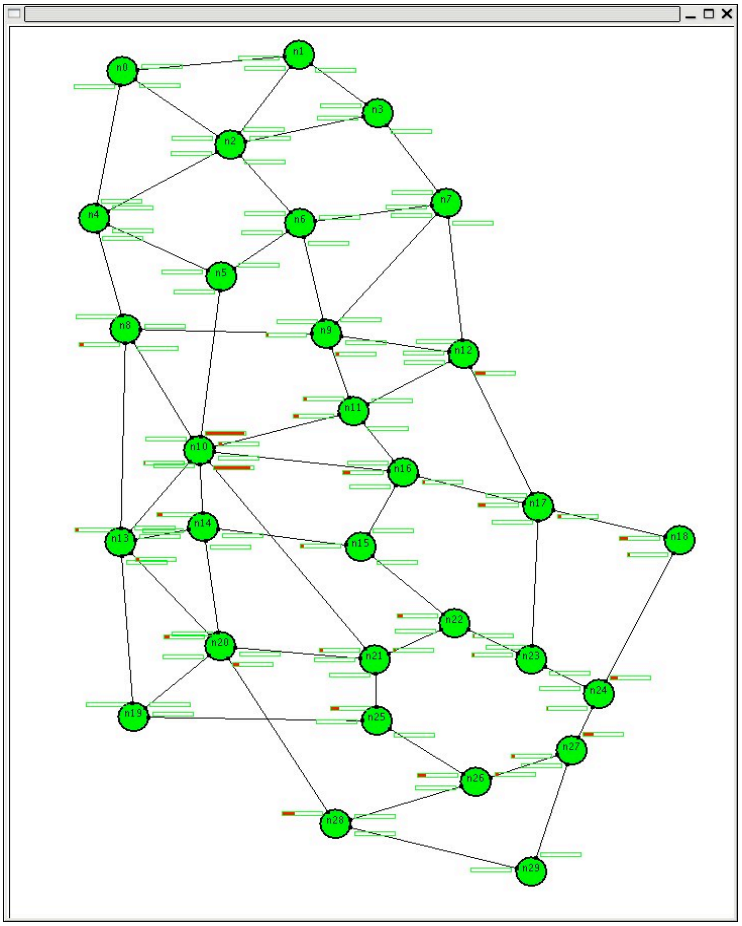

Fig. 10. An example of a stable state

One can observe on this example that, even if the situation is stable, not everything is perfect: the node n10 has two overloaded interfaces. Thanks the habituation phenomenon, the agent on node n10 considers that the situation is rather positive, and adopts a positive state.

6 Conclusions and perspectives

IP networks, and particularly the Internet, are supposed to be simple and robust. Integration of new differentiated services new functionalities implies adding new configuration layers. These new layers become quickly too complex, and require a huge amount of high-level knowledge, that cannot be provided by standard exchange protocols. As a consequence, new functionalities, not optimally configured, are either under-exploited or even not used at all.

Facing this result, we propose a simple and robust architecture of autonomic and collaborative agents, which role is to share local and situated information, in order to build a global distributed knowledge plane. In a distributed environment like a network, distributed computing approaches (from Artificial Intelligence) like multi-agent systems ideally fit. Sharing their situated views of the environment, agents are building locally a representation of their environment in order to decide elementary configuration actions. We assert that providing such high-level situated knowledge to the control mechanisms will open the way to new smart control heuristics. These heuristics shall be less deterministic, and more based on fuzzy/uncertain global knowledge.

We describe the application of our technology to a concrete issue: configuration of OSPF weights. To address this complex problem, our solution use a new set of heuristics, based on a situated representation of the local neighborhood (which is provided by our multi-agent system). This solution presents a convincing behavior. We manage to reduce drastically the number of overloaded interfaces, thanks to a simple, robust, scalable, and flexible new technology.

However, a lot of further studies are planned. As a first step, we will have to develop new test patterns. Even if qualitative tests provide appreciable information about the processes, we need to ensure that this technology allows us to get better performance within acceptable delays. We have to test our system with credible traffic models, and to observe the behavior of the system depending on the values of the parameters of the agents.

On another hand, new applications will allow us to try different techniques, in order to enhance our agent model. We particularly began to develop an alarm filtering and management application, which relies on more symbolic data and algorithms, when OSPF application was using more numeric data. We also improve with this new application new knowledge sharing mechanisms.

7 References

[FER99] Jacques Ferber, Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1999

[MES03] Nada Meskaoui, Leila Merghem, Dominique Gaiti, Karim Y. Kablan, Diffserv network control using a behavioral multi-agent system, January 2003

[MER03] Leila Merghem, Dominique Gaiti, and Guy Pujolle, On Using Multi-agent Systems in End to End Adaptive Monitoring, Lecture Notes in Computer Science, Springer-Verlag GmbH, Volume 2839 / 2003, Management of Multimedia Networks and Services, Chapter: pp. 422 – 435

[YE02] T Ye, HT Kaur, S Kalyanaraman, KS Vastola, S Yadav, Dynamic Optimization of OSPF Weights using Online Simulation – proceedings of IEEE INFOCOM, 2002

[FOR00] B Fortz, M Thorup, Internet traffic engineering by optimizing OSPF weights – PROC IEEE INFOCOM, 2000

[ERI01] M. Ericsson, M.G.C. Resende, and P.M. Pardalos, A Genetic Algorith for the Weight Setting Problem in OSPF Routing, Oct 2001

[BIE98] A Bieszczad, B Pagurek, T White, Mobile Agents for Network Management – IEEE Communications Surveys, 1998

[IBM03] AG Ganek, TA Corbi, The dawning of the autonomic computing era - IBM Systems Journal, 2003

[CIS98] T.M. Thomas, OSPF Network Design Solutions, Cisco Press, 1998

[WAN99] Y. Wang and Z. Wang, Explicit Routing Algorithms for Internet Traffic Engineering, proceedings of 8th Computer Communications and Networks, 1999

[BER92] D. Bertsekas and R. Gallager, Data Networks, Prentice Hall, 1992