Scalable Service Deployment on Software-Defined Networks

Javier Rubio-Loyola, CINVESTAV Tamaulipas
Alex Galis, University College London
Antonio Astorga and Joan Serrat, Universitat Politècnica de Catalunya
Laurent Lefevre, INRIA
Andreas Fischer, Alexandru Paler, and Hermann de Meer, University of Passau

ABSTRACT

It is widely accepted that the network of the future will require a greater degree of service awareness and optimal use of network resources. This article presents an architectural design for an open software-defined network infrastructure that enables the composition of fast and guaranteed services in an efficient manner and the execution of these services in an adaptive way, taking into account better shared network resources provided by network virtualization. Validation results are provided with special emphasis on service deployment scalability over virtualized network infrastructures.

INTRODUCTION

It is becoming accepted that future networks should be service- and management-aware [1], which includes (among others) the following aspects:
- Delivery of content and service logic with consumers' involvement and control
- Fulfilment of business, quality of service (QoS), and service level agreements (SLAs)
- Optimization of the network resources during service delivery
- Composition and decomposition on demand of control mechanisms and network domains

Conversely, deployed services [2] in the future networks should be network-aware. Network awareness means that the consumer-facing and resource-facing services are aware of the properties, requirements, and state of the network environment. This enables services to self-adapt according to changes in the network context.

In recent years, network virtualization techniques have gained a lot of attention due to their flexibility for creating computing clouds, and creating separate and independent virtual networks on top of physical network infrastructures.

Virtual networks abstract away the complexity of the underlying infrastructure. They are characterized in the literature as either the main means to test new network architectures or a crucial component of future networks [3–5]. Multiple logical networks can coexist above the same physical substrate. They can take the form of virtual private networks [6], active and programmable networks [7], overlay networks [8], or virtual networks [9]. The virtual nodes and links form a virtual topology over the underlying physical network.

Virtual networks are a collection of virtual nodes connected together by a set of virtual links to form a virtual topology, which is essentially a subset or an aggregation of the underlying physical topology. Virtual networks aim at better utilization of the underlying infrastructure in terms of:
- Reusing a single physical or logical resource for multiple other network instances
- Aggregating multiples of these resources in order to obtain more functionality, such as providing a pool of resources that can be utilized on demand

As an example, virtual networks can be aggregated (or federated) together. Such an approach requires aggregation and dissolution of control, data, and information planes, which is a challenging problem.

Virtualized network environments are highly dynamic [4] as links and nodes may be reconfigured quickly. Virtual routers may migrate on-demand, as in [3], based on resource availability, in order to save energy or follow the physical location of users. Nodes may also move logically (i.e., not physically). Virtual network aggregation or dissolution triggers changes in the virtual topology (i.e., virtual network embedding) [9]. Management of virtual networks is challenging, since it is necessary to manage this complex functionality. Manageability and service deployment are considered to be the biggest concerns for network virtualization [4].

This article describes the architectural model and validation results of the European Union (EU) Autonomic Internet (AutoI) project [10], which proposes open source software-defined networks as part of future networks [26]. It is a self-managing overlay of virtual resources that can span across heterogeneous physical networks. All of the components developed in the

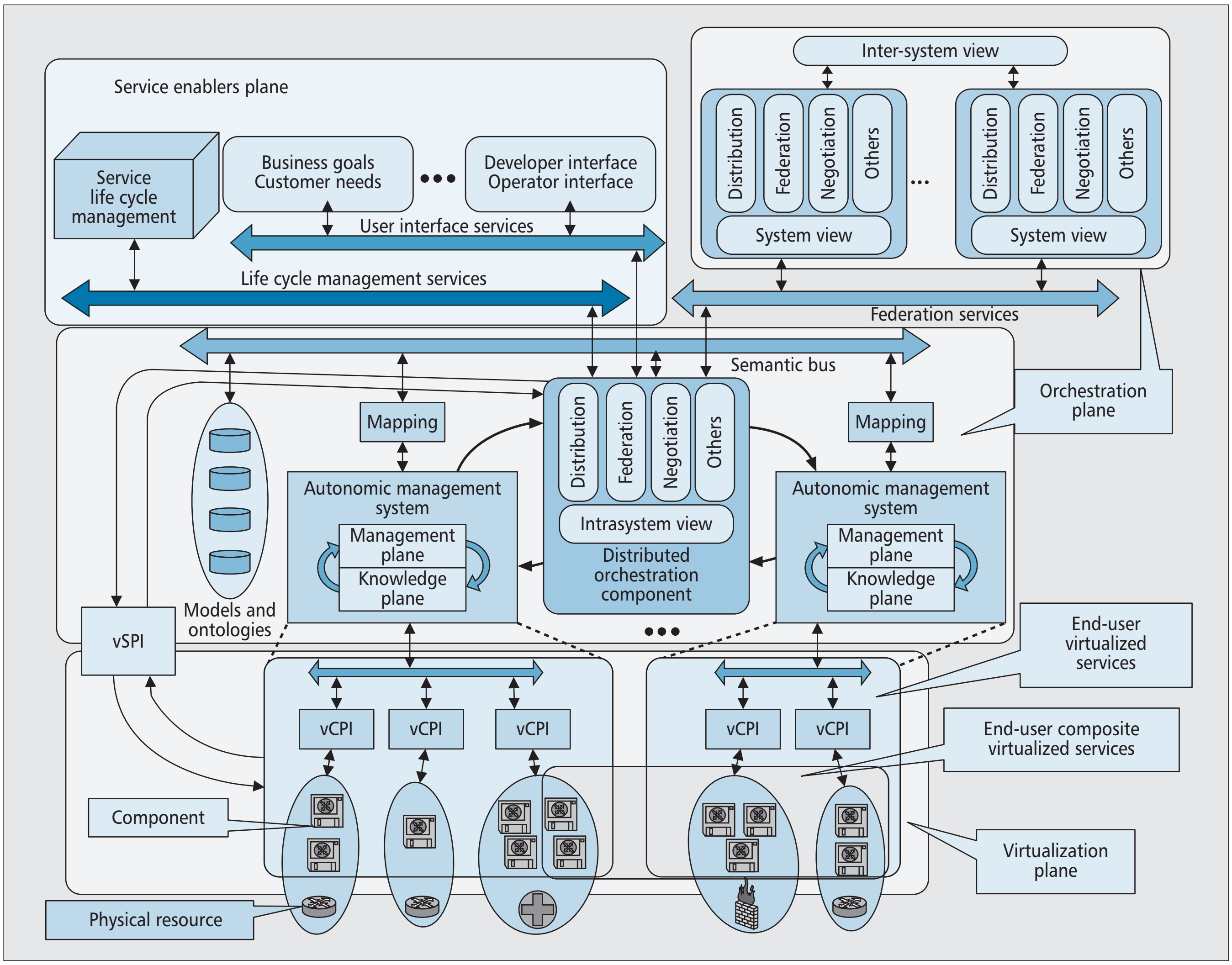

Figure 1. *AutoI architectural model: software-defined network*

AutoI architecture are available as open source components from [10]. Validation results were performed by exercising the open source components on three physical networks ranging from 4 to 5000 nodes. Two types of validation results based on large-scale experiments are the focus of this article as follows:

- Built-in network management, specifically self-management functionality for service and networking awareness
- Large-scale provisioning and deployment of both application services and management services over virtual infrastructures

This article is structured as follows. We present the AutoI architectural framework and its relevant systems. We present validation results. We provide some technical discussion. We describe related work. We then conclude the article.

AutoI Architectural Framework and Systems

The AutoI framework consists of a software-defined network described with the help of five abstractions, the OSKMV planes: orchestration plane (OP), service enablers plane (SP), knowledge plane (KP), management plane (MP), and virtualization plane (VP). At the physical level, they are embedded on network hosts, devices, and servers within the network. The main purpose of the OSKMV planes is to make future networks capable of self-knowledge and, ultimately, fully self-managing. The AutoI architectural model is shown in Fig. 1, and a description of the major elements follows. Open source platform components developed in support of the AutoI architecture are indicated by asterisks in the following section.

Orchestration Plane Overview

The purpose of the OP is to govern the behavior of the system in response to changing context, and in accordance with applicable business goals and policies. It supervises and integrates all other planes' behavior, ensuring integrity of the future Internet management operations.

The OP is a control framework into which any number of components can be plugged in order to achieve the required functionality. These components could have direct interworking with control algorithms, situated in the control plane of the Internet (i.e., to provide real-time reaction), and interworking with other management functions (i.e., to provide near-real-time reaction).

A key advantage of the AutoI architecture is that it can provide a programmable mix of isolation and sharing of network resources. A key advantage of separating the control and data planes is to provide increased isolation for an application or set of applications.

In practical terms, the OP controls one or more autonomic management systems (AMSs, described later). It acts as control workflow for AMSs ensuring their bootstrapping, initialization, dynamic reconfiguration, adaptation, contextualization, optimization, organization, and closing down. It is functionally integrated by one or more distributed orchestration components (DOCs) and a dynamic knowledge base consisting of a set of models and ontologies, and appropriate mapping logic and buses. DOCs can federate via buses. The internal design details of the OP can be found in [11].

Autonomic Management System

A key advantage of the AutoI architecture is that it can provide a programmable mix of isolation and sharing of network resources. A key advantage of separating the control and data planes is to provide increased isolation for an application or set of applications.

Each AMS is part of the MP (described later), and includes interfaces to a dedicated set of models and ontologies and interfaces to one or more DOC(s). Mapping logic enables the data stored in models to be transformed into knowledge and combined with knowledge stored in ontologies to provide a context-sensitive assessment of the operation of one or more virtual resources. Another set of interfaces enables framework services, such as directory services, naming, federation, and others, to be used by the AMS.

Distributed Orchestration Component

The DOC* provides a set of framework network services. Framework services provide a common infrastructure that enables all AMSs controlled by the OP to have (un) plug-and-play behavior. Applications compliant with these framework services share common security, metadata, administration, and management services. The DOC enables the following framework network services to the AMSs under its control:

Federation: Each AMS is responsible for its own set of virtual and non-virtual resources and management services that it governs. Federation enables a set of domains to be combined into a larger domain, where selected functionality of each constituent domain contributes to the overall functionality of the larger domain.

Negotiation: Each AMS advertises a set of capabilities (i.e., services and/or resources) that it offers for use by other components in the OP.

Distribution: This service enables tasks to be split into parts that run concurrently on multiple AMSs controlled by a DOC, or even across multiple DOCs. This function ensures that AMSs with different implementations and functionality can collaborate.

Governance: This service enables each AMS to be able to operate in an individual, distributed, or collaborative mode. Business goals, service requirements, context, capabilities, and constraints are all considered as part of the decision making process.

Intrasystem view: This service provides an overall composite view of the system as seen by the components within a DOC.

Intersystem view: this service provides an overall composite view of collaborating DOCs, as in a multdomain system.

Service Enablers Plane Overview

The S consists of functions for the automatic (re)deployment of new management services, protocols, and resource-facing and end-user-facing services. It includes enablers to allow code to be executed on the network entities. This functionality is implemented by the autonomic network programming interface (ANPI*) [10], which supports large-scale network programmability in deployed virtual networks. The safe and controlled deployment of new code enables new services to be activated on demand and made available to both management and orchestration planes for the benefit of service awareness. This approach has the following advantages:

- Automatic service deployment, allowing a significant number of new services to be offered on demand
- Flexible network configuration capabilities
- Special management functions and services easily enabled locally for testing purposes before they are automatically deployed network-wide
- Flexible support for service migration, for both consumer-facing and resource-facing services

Knowledge Plane Overview

The KP was proposed in [13] as a new dimension to a network architecture, contrasting with the data and control planes; its purpose is to provide knowledge and expertise to enable the network to be self-monitoring, self-analyzing, self-diagnosing, and self-maintaining.

AutoI introduces a narrow functionality KP, consisting of models and ontologies, to provide increased analysis and inference capabilities. AutoI's KP brings together widely distributed data collection, wide availability of that data, and sophisticated and adaptive processing or KP functions within a unifying structure. This brings order, and meets the policy, scaling, and functional requirements of a global network. The main KP components are a context and service information platform (CISP*) [10] and ontologies, which enable the analysis and inferencing capabilities. The CISP provides:

- Information life cycle management (storage, aggregation, transformations, updates, distribution) of all information and context in the network, addressing the Internet's size/scope
- Responsiveness to requests made by the AMS
- Triggers for the purpose of AMS contextualization
- Support for robustness, enabling the KP to continue to function as best possible, even under incorrect or incomplete behavior of the network itself
- Support of virtual networks and virtual system resources, enabling them to cooperate for mutual benefit in more effective network management

Management Plane Overview

The MP governs all virtual resources, performing decisions on their optimal placement, function and continuous migration. The functionality of the MP is implemented by the AMS (*). The

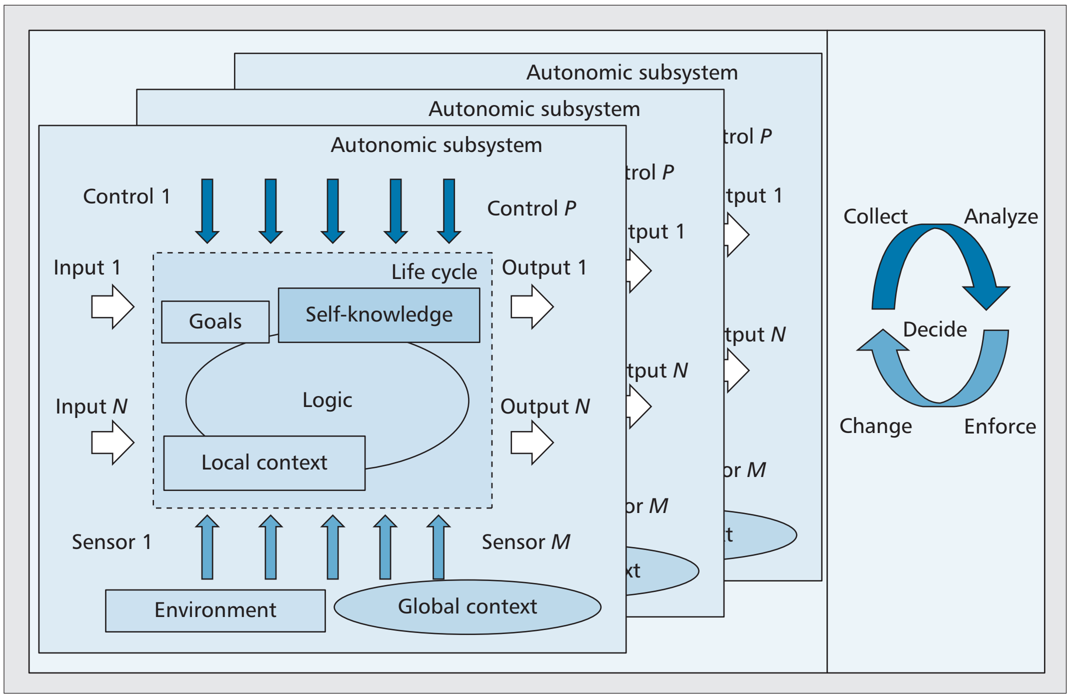

Figure 2. *Autonomic control loops.*

AutoI uses platform virtualization to provide virtual services and resources. Platform virtualization separates an operating system from its underlying platform resources; resource virtualization abstracts physical resources into manageable units of functionality.

MP functionality is subject to constraints determined by the OP. The MP is designed to meet the following objectives:

Embedded network functions: The majority of management functionality is embedded in the network and is abstracted from human activities. The AMSs run on execution environments on top of virtual networks and systems, which run on top of all current network (i.e., fixed, wireless and mobile networks) and service physical infrastructures.

Aware and self-aware functions: It monitors the network and operational context as well as internal operational network state in order to assess if the network current behavior serves its purposes.

Adaptive and self-adaptive functions: It triggers changes in network operation (state, configurations, etc.) as a result of changes in network and service context.

Automatic self-functions: It enables self-control of its internal network operations, functions, and state. Manual/external input is provided in the setting up of business goals and other unavoidable functions.

Extensibility functions: It adds new functions without disturbing the rest of the system (plug-and-play dynamic programmability of management functions and services). The AMSs are designed to follow the autonomic control loops (collect, analyze, decide, enforce, change [14]) depicted in Fig. 2.

Virtualization Plane Overview

One of the key requirements that differentiate AutoI from other systems is its emphasis on virtualization of resources and services. AutoI uses platform virtualization to provide virtual services and resources. Platform virtualization separates an operating system from its underlying platform resources; resource virtualization abstracts physical resources into manageable units of functionality. For example, a single physical resource can appear as multiple virtual resources (e.g., the concept of a virtual router, where a single physical router can support multiple independent routing processes by assigning different internal resources to each routing process); alternatively, multiple physical resources can appear as a single physical resource (e.g., when multiple switches are "stacked" so that the number of switch ports increases, but the set of stacked switches appears as a single virtual switch).

AutoI extends contemporary virtualization approaches and aims at building an infrastructure in which virtual machines can be dynamically relocated to any physical node or server regardless of location, network, storage configurations, and administrative domain.

The VP consists of software mechanisms to treat selected physical resources as a programmable pool of virtual resources that can be organized by the OP and MP into appropriate sets of virtual resources to form components (e.g., increased storage or memory), devices (e.g., a switch with more ports), or even networks. Two special interfaces, called the virtualization system programming interface (vSPI, described below) and virtualization component programming interface (vCPI), assess the basic functionality of the VP.

Virtualization System Programmability Interface

The vSPI contains the "macro-view" of the virtual resources a particular OP governs. The low-level configuration (i.e., the "micro-view") of a virtual resource is provided by the vCPI, as explained in the next section.

The vSPI* is responsible for determining what portion of a component (i.e., set of virtual resources) is allocated to a given task. This

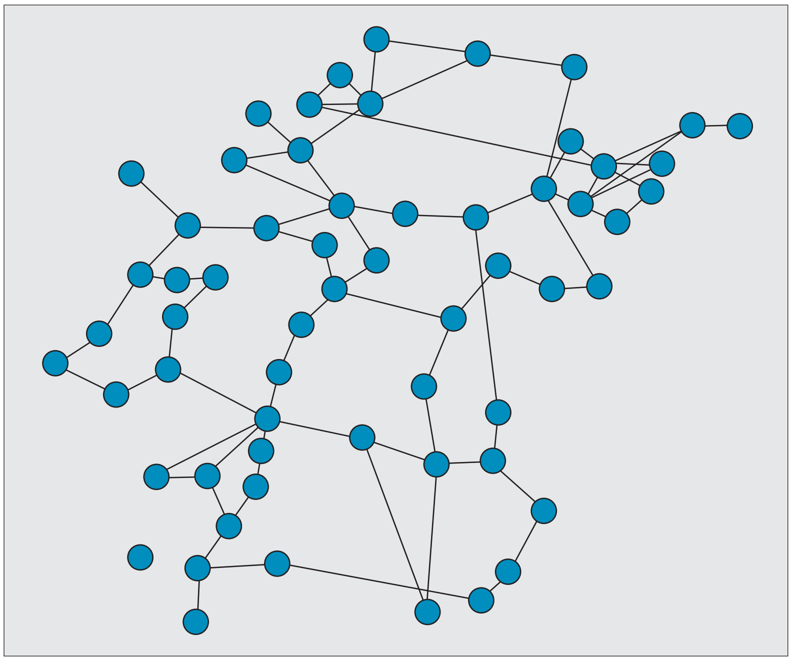

Figure 3. *60-virtual-router and 80-virtual-link virtual network created in a four-component physical testbed.*

means that all or part of a virtual resource can be used for each task, providing an optimized partitioning of virtual resources according to business need, priority, and other requirements. Composite virtual services can thus be constructed using all or part of the virtual resources provided by each physical resource.

The vSPI monitors the "macro-level" status of the virtual resources it governs. This is different from the vCPI, which monitors "micro-level" status of the virtual resources it configures. For example, the vSPI collects global information about available physical resources. When an AMS receives requests to instantiate a new virtual resource, it contacts the vSPI to determine which physical resources can be used. The vSPI then instructs the vCPI to instantiate the virtual resources on the correct physical resource for such purpose. The vSPI informs the AMS when the virtual resource is ready for use, and the vCPI informs the AMS when each virtual resource has been successfully reconfigured.

VIRTUALIZATION COMPONENT PROGRAMMING INTERFACE

The vCPI* is a modular, scalable, and communication-protocol-agnostic system for monitoring and managing virtual resources. It operates locally; for each component of a physical network, there is an embedded vCPI that operates with third-party software by using a request/response mechanism. It is used for constructing, modifying, and managing virtual networks (VNs) consisting of virtual links (VLs), virtual routers (VRs), and routing services (RSs). This enables the AMS to manage the physical resource, and to request virtual resources to be constructed from that physical resource via the vCPI. The AMS sends device-independent commands to the vCPI, which are translated into device- and vendor-specific commands that reconfigure the physical resource and manage the virtual resources provided by that physical resource. The vCPI also provides monitoring information from the virtual resources back to the AMS which controls that physical resource. Note that the AMS is responsible for obtaining management data describing the physical resource.

The vCPI is responsible for providing dynamic management data to its governing AMS, which states how many virtual resources are currently instantiated and how many additional virtual resources of what type can be supported. The vCPI needs to be aware of the structural information of the relations among virtual resources; therefore, a discovery mechanism is included to inspect the contents of a physical component and map it to a data structure.

More details of the AutoI open source software defined network components that integrate the five-plane AUTOI approach introduced earlier are provided in [10, 12].

PRACTICAL APPROACH AND VALIDATION RESULTS

This section tests the AutoI framework and the open source components, which were installed and run on three physical testbed networks ranging from 4 to 5000 nodes. It provides relevant results emphasizing the creation of virtual networks, service deployment, and scalability aspects. This section concentrates on wired physical networks. The interested reader will find a description of the AutoI support for wireless networks in [10].

CREATION OF VIRTUAL NETWORKS

In AutoI, virtual networks can be set up by means of administrative decisions, programmed at a given time, triggered by events like threshold crossings, or they can be created on demand.

This section demonstrates the AutoI context-aware, on-demand, scalable creation of functional virtual networks. The term *functional* implies that virtual networks are ready to support application services deployment, having all networking services (e.g., routing services) deployed and configured properly. For this purpose we have used a physical infrastructure consisting of four physical components (2 Quad Core AMD Opteron 2347H CPUs and 32 Gbytes RAM, running Linux, XEN, or Qemu and Open SSH).

In this test setting the autonomic loop's collect part (see Fig. 2 for details of this loop) is exercised by the Context Information Services Platform (CISP) that manages context information and triggers notifications about changes corresponding to demand requests for virtual networks. Demand request correspond to end-points of connectivity with specific QoS constraints (e.g., throughput). CISP nodes are deployed in each physical node and also in each virtual router as soon as it is created. The collected data is made available to all AutoI components along the life cycle of the virtual infrastructure.

The autonomic loop's *analyze* and *decide* parts are implemented by the AMSs and DOCs, which analyse context changes (e.g., endpoint network requests) and evaluate the conditions under which context changes occur, and eventually decide on appropriate actions that would fulfill their business goals. Business goals in this test case correspond to specific characteristics of the to-be-created virtual network topology. In this test scene a network topology that emulates the current (2010) topology of the German X-WiN network [15] was used. Such a network consists of 60 nodes interconnected by 80 links, representing sites all over Germany as graphically depicted in Fig. 3. The virtual routers are spread among the four physical components, 15 VRs are created on each component. Decisions are taken and result in concrete configurations that will need to be enforced via the vSPIs, vCPIs, and ANPI platforms.

The autonomic loop's enforce parts are assessed by the vSPIs, vCPIs, and ANPIs. They are aimed at executing the appropriate commands to configure the VN. In practical terms, creating a VN is a two-step process:
• Creation/startup of VRs and creation of VLs attached to the former (*enforced* by the vSPIs and vCPIs)
• Deployment of the networking-facing services (e.g., RSs) that would support the VN (enforced by the ANPIs)

A VL exists between two VRs and consists of three segments: one is connecting the physical hosts, and the other two are connecting each VR with its host. All three segments are aggregated to a VL by two software bridges (driven by two vCPIs). Therefore, for each VL (e.g., between VR1 and VR2) the following operations are conducted:
• Create the central segment (tunnel) between the physical components (optional if the linked routers are hosted on the same physical component)
• Add a virtual network interface to both VR1 and VR2
• Bridge each virtual interface to the corresponding physical component to create the first link segments
• Deploy the networking facing services in VR1 and VR2

To start up a VR, the vCPI uses hypervisor commands to create a VR from a template image. This template is instantiated with individual configuration options, like the initial network address or the amount of virtual hardware to be assigned. The tunnels to instantiate VLs are created using OpenSSH. Tunnels that are using the same physical network interface card (NIC) share the available bandwidth among themselves. Statically assigning a guaranteed amount of bandwidth for a VL is possible by using traffic control mechanisms as demonstrated in [16].

The AMSs and DOCs in this test scenario issue commands to the vCPIs sequentially, which in turn have processed them (in parallel when possible). The startup times for the VRs are depicted in Fig. 4, where the most relevant behavior is that the startup times are stable for an increasing number of VRs in each physical component. In this execution run each VR needs on average 25 s to be started. After the routers are started, the AutoI solution reacts to this context change (new virtual resources available) and issue appropriate commands to the vCPIs to

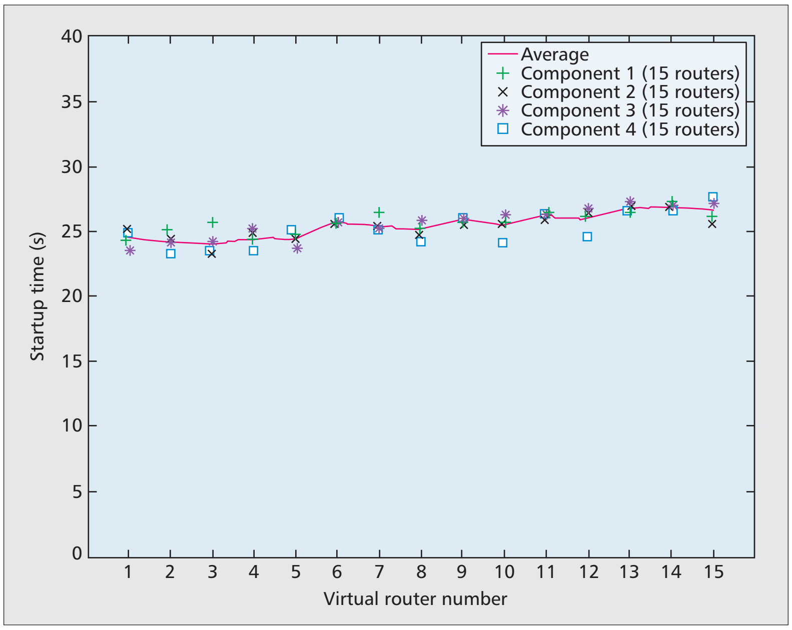

Figure 4. *Startup times of 60 virtual routers created in a four-component physical testbed.*

instantiate the VLs. The vCPIs enforce the commands in the four components of our testbed in parallel when possible, using the three-segment approach described earlier. Again, the most relevant behaviour is that the startup times of each VL are stable for an increasing number of VLs in the four physical components. The average time to construct a VL is about 5 s with a little variation between the minimum and maximum values as graphically depicted in Fig. 5.

As mentioned earlier, the autonomic loop's enforce parts assessed by the ANPIs are devoted to discover and deploy the networking-facing services (i.e., routing) that will make such network operational. The round-trip times between each VR were measured to test that the network is operational. The minimum, maximum, and mean values of five measurements are presented in Fig. 6. The most important behavior here is that the average round-trip time appears to be linear with the number of virtual hops, with a moderate slope and values below 40 ms in our testbed.

Service Deployment Results

This section demonstrates the AutoI context-aware on-demand service deployment capabilities over VNs. As in our last test scene, the autonomic loop's *collect* part is exercised by the CISP that manages context information and triggers notifications about changes corresponding to service requests for VNs, availability of services, resource usage information, and so on.

The autonomic loop's analyze and decide parts are implemented by the AMSs and DOCs, which analyze context changes (e.g., service requests) and evaluate the conditions under which context changes occur, and eventually decide on appropriate actions that would fulfill their business goals. Business goals in this test case would have an impact on the level of service deployment that would be eventually enforced. For example, local service support implies that a service would be deployed in a single virtual element, whereas domain or global support would imply that a set of services be deployed by network operators to a specific area of the VN.

The autonomic loop's enforce part is assessed by the ANPI, which executes a number of atomized autonomic processes in each node of the network when necessary. The main AutoI components enforcing this service deployment test are graphically depicted in Fig. 7.

The ANPI maintains service repositories with available tested service code. Services are discovered by the ANPIs and are available for download from repositories located in the VN.

ANPI daemons deploy and manage the life cycle of services. There is one ANPI running in each virtual component of the deployed network. When the ANPI receives a command from an AMS and/or DOC component to deploy or migrate a specific consumer-facing and/or resource-facing service, it analyzes such information, and this triggers a new decision making process in the ANPI nodes. The type of service, characteristics, availability of services in the network, and so on is information that is taken into account to decide the best deployment steps.

The ANPI communicates through the CISP to expose and notify service deployment operations and services states.

Following on the described test case, the action of a client requesting a streaming service in the network of Fig. 7 is emulated. AutoI creates a new VR on demand and attaches it to the new client. The ANPI discovers the location of the appropriate streaming services to drive the configuration of the required networking services (e.g., routing services in this case) and supporting services (e.g., context, monitoring) required. In each router of the network (now with eight VRs), the ANPI deploys a basic routing service. Figure 8 shows the time taken (bottom part) for the deployment of 8 ANPIs (continuous line), and the time taken (dotted line) to deploy 12 services (network-facing and application-facing) required to provision an end-to-end streaming service over the deployed virtual infrastructure. It is worth mentioning that once all ANPIs are deployed, there is a small gap of time taken by the AutoI systems to find and download the required services and correlate appropriate context changes. After this, the services are deployed, configured, and started in about 3 min, as depicted on the right of Fig. 8.

Physical resources have limited capacities in terms of bandwidth, CPU, memory, and so on, which in turn are shared among virtual resources. Lack of physical resources is eventually manifested as service degradation. AutoI reacts effectively to service degradation with coordinated service migration actions, in which all virtual networking services and application services are actually reconfigured in appropriate physical resources. The interested reader will find extensive results of the AutoI migration support in [10]. The remainder of the article focuses on scalability support.

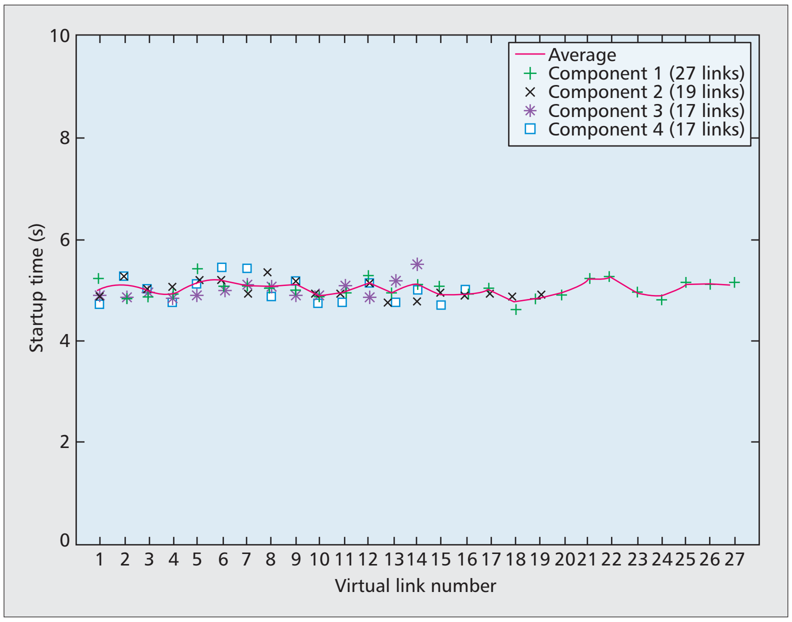

Figure 5. *Virtual link creation times in a four-component physical testbed.*

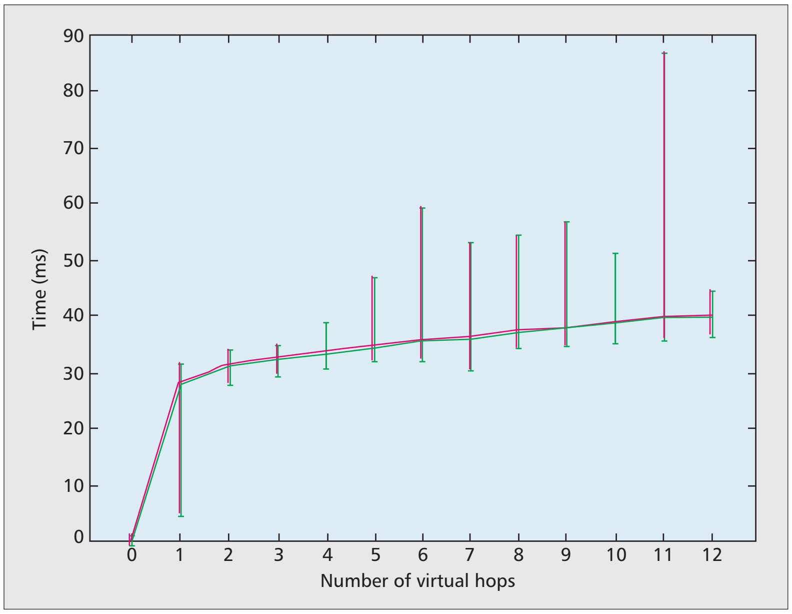

Figure 6. *Round-trip times in 60-router/80-link virtual network*

Scalability Results on Network Virtualization and Service Deployment

Large-scale validations were performed in an experimental testbed (Grid5000 testbed [17]) composed by a cluster of 10 separate sites located in France, where all AutoI components were installed. The testbed supports 5000 cores located on various clusters connected with 10G links.

This section analyzes the scalability of the AutoI solution for network virtualization and service deployments on a large scale, in similar setups as presented for mid-scale validations earlier.

Table 1 shows the result of the deployment of 10–150 VRs on 10–50 physical machines (PMs) on two sites of the testbed. It is worth mentioning that the deployment of virtual machines depends on the number of VRs to deploy more than on the number of PMs. Observed results are closely similar between Grid5000 sites. Table 1 shows the results obtained from two sites (Bordeaux and Lyon).

For service deployment tests, target topologies like chains or trees were generated with the means to allow analysis of observed results. Figure 9 presents the service deployment results of a virtual network with 110 VRs in a chain topology in the Grid5000 testbed. The deployment of 110 ANPI daemons with its activation occurred in 30 s (continuous line in Fig. 9), while the deployment of 110 small sized services (4.4 kbytes) required only 8 s (dotted line in Fig. 9).

Figure 10 presents the results of 220 services deployment on 110 VRs located in one VN with tree topology. This scenario shows that service deployment can occur at any time during the life of components provided that the ANPIs are already deployed in each VR. The deployment of the first 110 services occurred in 8 s (first part of the dotted line in Fig. 10). After a small gap, the remaining 110 services were progressively deployed in about 9 min.

Technical Discussion and Lessons Learned

One of the primary arguments for setting up multiple virtual infrastructures is the possibility to support networks with different network protocols on the same hardware. However, it is necessary to develop and provide the means to also manage the virtual network elements, in particular the configuration of the virtual network layer. In order for network virtualization techniques to be a key component for the next-generation Internet, network virtualization interfaces (like the vCPI and vSPI in our architectural model) need to be able to configure virtual network interfaces, while at the same time remaining protocol-agnostic and open to future Internet protocol stacks.

Stability of virtual networks is an important issue that deserves special attention. Several instability problems were encountered during our experimental research. For example, when trying to create a number of coexisting virtual networks, each with six virtual routers distributed over four physical components, instability problems occurred when going beyond 15 virtual networks. Beyond this point, the creation of both virtual links and virtual routers became unpredictable. From our numerous experiments the conclusion drawn is that current virtualization technologies are not built to cope with the dynamicity and load expected in future virtualized network environments. Additional implementation effort is needed to bring both hypervisor and virtual link technologies to a level where arbitrary creation of virtual networks becomes possible.

Programmability in network and services encompasses the study of decentralized enablers for dynamic (de)activation and reconfiguration of new/existing services, including management services and network components. AutoI has taken the challenge to enable trusted parties (users, operators, and service providers) to activate management-specific service and network components into a specific platform. Dynamic programming enablers will be created as executable service code, which can be injected/activated into the system's elements to create the new functionality at runtime. Network and service enablers for programmability can therefore realize the capabilities for flexible management support.

Large-scale validation experiments were performed to validate the efficient support of AutoI service deployment support. Due to the high quality (in terms of latency and throughput) of the testbed, the time required to deploy services was extremely short. However, deploying services in less reliable infrastructures requires more fault-tolerant approaches.

Deployment of VR	10PM * 1VR = 10 VR	10PM * 3VR = 30 VR	50PM * 1VR = 50 VR	50PM * 3VR = 150 VR
Lyon	55 s	3 min 18 s	1 min 38 s	3 min 42 s
Bordeaux	57 s	3 min 7 s	1 min 14 s	4 min 41 s

Table 1. *Virtual infrastructure deployment from 10 to 150 VRs.*

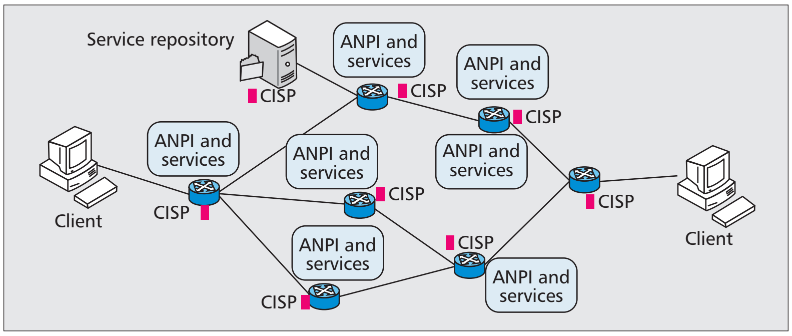

Figure 7. *Basic virtual network with AutoI service deployment support.*

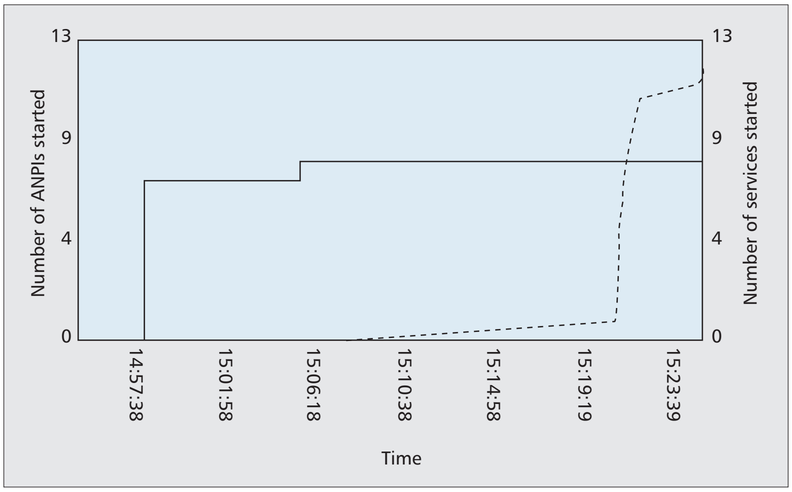

Figure 8. *Deployment of ANPIs (solid) and deployment of services (dotted).*

Related Work

The last decade has seen tremendous interest in all aspects of the future Internet (FI). As a comprehensive survey would require more than a single article, this section describes key initiatives in the United States and Europe contributing to the development of the FI.

From the U.S. perspective, the National Science Foundation (NSF) supports four big projects. Named Data Networking [18] is aiming at an

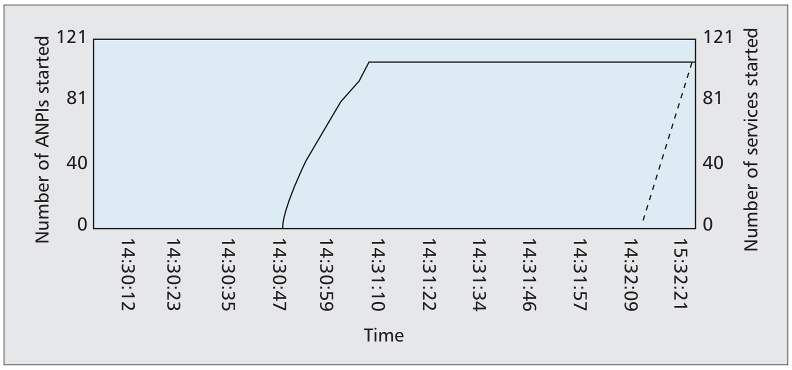

Figure 9. *Deployment of 110 ANPIs (solid) and 110 services (dotted).*

approach to identify the content to be supported by the future networks by itself instead of the locations where it resides. Mobility First [19] is looking at the inherent challenges of mobility and, in particular, the use of opportunistic networking to support communications between end points. NEBULA [20] is aiming at a cloud computing architecture as a means to guarantee always available services. Finally, eXpresive Internet Architecture [21] addresses the growing diversity of network use models, the need for trustworthy communication, and the growing set of stakeholders who coordinate their activities. The scope of all these projects is much broader and concentrated on issues different than the ones presented in this article. None of them use an orchestration plane to coordinate autonomic systems and do not explicitly mention the autonomic networking paradigm in support of service deployment and maintenance.

Closely related to our objectives, we can mention the TRILOGY, SOCRATES, 4WARD, and UniverSelf EU projects.

The focus of TRILOGY [22] is on the development of the generic control functions of the network. These control functions deal with routing mechanisms, resource control, and social and commercial control. Instead, our approach concentrates on a management plane in support of service deployment.

The scope of the SOCRATES project [23] is the bottleneck problems created by the mobile access network and proposes self-* mechanisms to create a solution. In that sense our approach adopts the same conceptual solution because the autonomic Internet as we have conceived it has to make extensive use of self-* mechanisms. Nevertheless, the AutoI autonomic approach goes beyond the access network and makes it an integral part of the fixed network as well.

4WARD [24] makes use of paravirtualization systems like the AutoI approach to virtualize routers and network links. The most important similarity is the management approach: both approaches adopt the autonomic paradigm. Nevertheless, the main difference is in the architectural approach. 4WARD highlights the "in-network management" solution, that is, embedding the management functionality in the same managed network devices. Instead, AutoI creates a multilevel hierarchy involving the service plane, the distributed orchestration elements, and the autonomic management systems, which take care of one or many devices as necessary. Separating the management activities in planes and not associating management functions with managed devices makes our solution scalable and easy to deploy in current network infrastructures.

UniverSelf [25] is meant to create a framework federating different self-management approaches to make the FI a global autonomic management system.

Concluding Remarks

This work has presented the design and validation results of an open software-defined network infrastructure (i.e., a network cloud) that enables fast and scalable composition of services in an efficient manner, and the execution of these services in an adaptive way, taking into account better shared network resources provided by a virtualized network substrate.

Current communication networks are composed of a set of heterogeneous resources. Virtualizing resources have served two purposes: managing the heterogeneity through introduction of homogeneous virtual resources and enabling programmability of central network elements. The flexibility gained through this approach helps to adapt the network dynamically to both unforeseen and predictable changes in the network.

The Autonomic Internet service deployment approach has demonstrated that dynamic programming can be used to enable creating new functionality at runtime over virtual infrastructures. Executable service code can be injected and activated into the virtual systems elements at runtime to give a higher degree of flexibility in the deployment of services in future networks.

In addition, the adopted approach has revealed its scalability when large-scale testbeds like Grid 5000 are used. Nevertheless, scalability in such a complex and multitier system as AutoI requires much more extensive testing than the experiments reflected in this article. We mean, for instance, experiments within scenarios stressing specific planes or components of planes. Considering the AutoI architecture, this would likely yield tens of scenarios. This article reflects the results of scalability tests in one particular scenario. Then we have to emphasize that our system scales well under the conditions of this particular scenario, and that by no means can these results be generalized to different scopes or situations. Additional testing is in fact part of challenging future work.

Acknowledgment

This work was undertaken in the context of the FP7-EU AUTOI project and MCYT TEC2009-14598-C02-02. Large-scale testing was performed in the Grid 5000 platform, supported by the French Ministry of Research. Special thanks to A. Cheniour, O. Mornard, and S. Clayman for their support.

References

[1] ITU-T Rec. Y.3001, "Future Network Vision —Objectives and Design Goals," http://www.itu.int/en/ITU-T/focus-groups/fn/Pages/Default.aspx

[2] B. Rochwerger *et al.*, "An Architecture for Federated Cloud Computing," *Cloud Computing: Principles and Paradigms*, R. Buyya *et al.*, Eds., Wiley, 2011.

[3] Y. Wang *et al.*, "Virtual Routers on the Move: Live Router Migration as a Network-Management Primitive," *ACM Conf. Data Commun. SIGCOMM 2008*.

[4] N. M. K. Chowdhury, R. Boutaba "A Survey of Network Virtualization," *Computer Networks*, vol. 54, Apr. 2010, pp. 862–76.
[5] T. Anderson *et al.,* "Overcoming the Internet Impasse Through Virtualization," *Compute*r, vol. 38, Apr. 2005, pp. 34–41.
[6] L. Andersson and T. Madsen, "Provider Provisioned Virtual Private Network (VPN) Terminology," IETF RFC 4026, Mar. 2005.
[7] A. Galis *et al.*, *Programmable Networks for IP Service Deployment*, Artech House, 2004.
[8] N. M. K. Chowdhury and R. Boutaba "Network Virtualization: State of the Art and Research Challenges," *IEEE Commun. Mag.*, vol. 47, no. 7, 2009
[9] M. Yu *et al.*, "Rethinking Virtual Network Embedding: Substrate Support for Path Splitting and Migration," *ACM SIGCOMM Comp. Commun. Rev.*, vol. 38, no. 2, 2008, pp. 17–29.
[10] EU IST Autonomic Internet Project web site: http://istautoi.eu.
[11] D. F. Macedo *et al.*, "The Autonomic Internet Approach for the Orchestration of Next-Generation Autonomic Networks," *J. Annals of Telecommun.*, Apr. 2011.
[12] J. Rubio-Loyola *et al.*, "Platforms and Software Systems for an Autonomic Internet," *IEEE GLOBECOM 2010*, Miami, FL, 2010.
[13] D. D. Clark *et al.*, "A Knowledge Plane for the Internet," *IEEE SIGCOMM 2003*.
[14] AG Ganek, TA Corbi "The Dawning of the Autonomic Computing Era," *IBM Sys. J.*, 2003, vol. 42, no. 1, pp. 5–18.
[15] J. Pattloch *et al.*, "X-WiN: The New German National Research and Education Network" *Praxis der Informationsverarbeitung und Kommunikation*, vol. 29, issue 1, pp. 50-53.
[16] A. Berl *et al.*, "Using System Virtualization to Create Virtualized Networks," *J. Electronic Commun. EASST*, vol. 17, 2009, pp. 1–12.
[17] Grid5000 TestBed www.grid5000.fr/
[18] Named Data Networking, http://www.nameddata.net/index.html
[19] Mobility First, http://mobilityfirst.winlab.rutgers.edu/
[20] Nebula, http://nebula.cis.upenn.edu/
[21] eXpressive Internet Architecture, http://www.cs.cmu.edu/~xia/
[22] EU IST FP7 TRILOGY Project "Re-Architecting the Internet: An Hourglass Control Architecture for the Internet, Supporting Extremes of Commercial, and Social and Technical Control" http://www.trilogy-project.org/
[23] EU IST SOCRATES project "Self-Optimisation and self-ConfiguRATion in wirelEss networkS" http://www.fp7-socrates.org
[24] EU IST FP7 4WARD Project, http://www.4ward-project.eu/
[25] EU IST FP7 UNIVERSELF Project "Realizing Autonomics for Future Networks" http://www.univerself-project.eu
[26] ITU-T Rec. Y.3001 "Future Networks: Requirements and Design Goals," http://www.itu.int/rec/T-REC-Y/e

BIOGRAPHIES

JAVIER RUBIO-LOYOLA (jrubio@tamps.cinvestav.mx) is a research scientist at CINVESTAV Tamaulipas, México. He holds an Engineering degree in communications and electronics, and an M.Sc. degree in digital systems, both from Instituto Politécnico Nacional de México, and a Ph.D. in telecommunications from Universitat Politècnica de Catalunya (UPC) in Barcelona, Spain. He has participated in a number of Spanish and IST-European research projects, mainly in the network management area. His research interests focus on network and service management, autonomic networking, and service engineering.

ALEX GALIS (a.galis@ee.ucl.ac.uk) is a visiting professor at University College London. He has co-authored seven research books and more than 150 publications in journals and conferences in the Future Internet areas: networks, services, and management. He has acted as PTC chair of 14 IEEE conferences and a reviewer for more than 100 IEEE conferences (www.ee.ucl.ac.uk/~agalis). He was the technical coordinator of the AutoI project.

ANTONIO ASTORGA (aastorga@nmg.upc.edu) is pursuing his Ph.D. in telecommunications at UPC. He holds an Engineering degree in communications and electronics from Instituto Politecnico Nacional of Mexico. Since 2008 he has been a research assistant of the Management, Pricing and Services in Next Generation Networks group (MAPS) chaired by Prof. Joan Serrat at UPC. He worked on the EU FP7 project AutoI. His research interests focus on autonomic networking and network management.

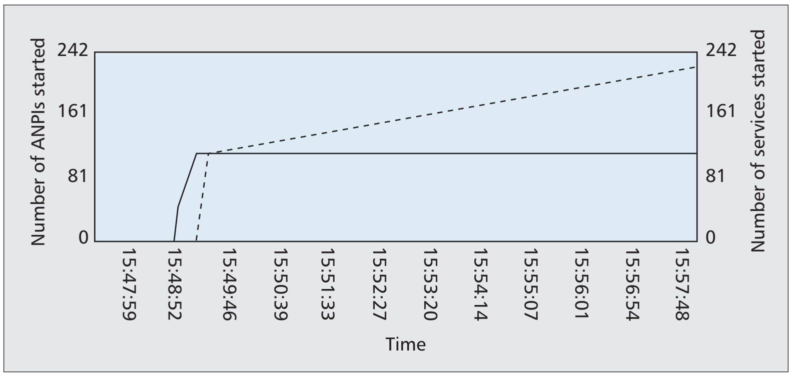

Figure 10. *Deployment of 110 ANPIs (line) with 220 services (dotted).*

JOAN SERRAT-FERNANDEZ (serrat@tsc.upc.edu) received his degree of Telecommunication Engineer in 1977 and his doctoral degree in telecommunication engineering in 1983, both from UPC. Currently he is a full professor at UPC where he has been involved in several collaborative projects with different European research groups, through both bilateral agreements and participation in European funded projects. His topics of interest are in the field of autonomic network and service management.

LAURENT LEFEVRE (laurent.lefevre@ens-lyon.fr) obtained his Ph.D. in computer science in January 1997 at LIP Laboratory (Laboratoire Informatique du Parallelisme) in ENS-Lyon (Ecole Normale Superieure), France. He is a permanent researcher in computer science at INRIA (the French Institute for Research in Computer Science and Control). He is a member of the RESO team (High Performance Networks, Protocols and Services) if the LIP laboratory in ENS. He has co-authored more than 80 papers published in refereed journals and conference proceedings. His interests include grid and distributed computing and networking, green and energy-efficient computing and networking, autonomic networking, and high-performance networks protocols and services.

ANDREAS FISCHER (andreas.fischer@uni-passau.de) received his computer science diploma from the University of Passau, Germany, in 2008. Since then he has been a scientific assistant of the Computer Networks and Computer Communications research group chaired by Prof. Hermann de Meer. His main research interests include autonomic networking, network virtualization, and resilience of virtualized environments. He worked on the EU FP7 project AutoI and is currently involved in the EU FP7 project ResumeNet. He is a member of the EuroNF Network of Excellence.

ALEXANDRU PALER (alexandru.paler@uni-passau.de) is pursuing his Ph.D. as a research assistant at the University of Passau. He graduated in computer science from Transilvania University of Brasov, Romania, and obtained his M.Sc. degree at the University of Applied Sciences Wiesbaden, Germany. His research focused on distributed systems and network virtualization. Currently he is investigating fault detection and diagnosis methods for quantum circuits.

HERMANN DE MEER (demeer@fmi.uni-passau.de) is a full professor of computer science (Chair of Computer Networks and Communications) at the University of Passau, Germwhere he is director of the Institute of IT Security and Security Law (ISL). He has been an assistant professor at Hamburg University, Germany, a visiting professor at Columbia University, New York, a visiting professor at Karlstad University, Sweden, a reader at University College London, United Kingdom, and a research fellow of Deutsche Forschungsgemeinschaft (DFG). He chaired one of the prime events in the area of quality of service in the Internet, IWQoS 2005, Passau. He has also chaired the first international workshop on self-organizing systems (IWSOS 2006, Passau) and the first international conference on energy-efficient computing and networking (e-Energy 2010, Passau).