Horizon Project

ANR call for proposals number ANR-08-VERS-010
FINEP settlement number 1655/08

Horizon - A New Horizon for Internet

WP1 - TASK 1.2: Choice of the Context-Aware Technology

Report

(Annex C)

Institutions

Brazil
GTA-COPPE/UFRJ
PUC-Rio
UNICAMP
Netcenter Informática ltda.

France
LIP6 Université Pierre et Marie Curie
Telecom SudParis
Devoteam
Ginkgo Networks
VirtuOR

Contents

List of Figures

Chapter 1

Introduction

Current Internet architecture has a unique protocol stack, the TCP/IP stack, running over the physical substrate. Thus, packets from all applications, each one with different performance requirements, are forwarded according to a single model with no differentiation. To provide customized services for the different applications in the future Internet, Horizon project proposes a pluralist architecture, which allows different protocol stacks running simultaneously over the same shared physical substrate. The proposed architecture is based on the concept of network virtualization [2]. The idea is to run multiple virtual networks on the same substrate providing resources for each network [3] and, thus, we are able to create different networks, according to the requirements of applications running on each virtual network.

For the proposed architecture, the efficient sharing of the underlying network resources is a fundamental challenge. Horizon project is developing the infrastructure required for a piloting system to deal with the resource-allocation problem among the different virtual networks. The goal is to project the needed information from the infrastructure onto the piloting system. Based on such information the piloting system acquires the knowledge needed to optimize the creation and destruction of virtual networks and the distribution of the physical resources. For each network element, such as routers and switches, the piloting system defines a situated view that is used to determine the context surrounding the element and to select and optimize their control algorithms and parameters. The main idea is to define local algorithms, such as routing and quality-of-service mechanisms, as a function of the context in order to increase network scalability. Hence, the piloting system must be aware of the context.

Context-aware systems are defined as systems that can be aware of the situation of entities or their contexts and do things for them [4]. According to this definition, context and situation are closely related concepts and both

are fundamentals to understand the context-aware systems. We assume that context is "any information that can be used to characterize the situation of an entity", as defined by Loke [4]. Location, time, computational resources, and network bandwidth are examples of context information. In addition, we assume that situation means a description of the current states of a given entity, as also defined by Loke [4]. For example, a link with slow responsiveness, high packet loss rate, and high delay (context information) is probably congested (situation). In most of the cases, as occurs in the previous example, we have to aggregate context information in order to determine the situation of an entity.

Context-aware systems must be able to determine what situations an entity are in and to detect changes of situation. Context awareness also enables the system to act automatically, which allows network control without human interference. In order to provide these functionalities, first, the system has to monitor the environment to acquire context information surrounding a network element. After that, it has to reason about the context acquired and then it has to act in order to achieve a goal. Entities have also to communicate to another to determine their situations in time. To tackle this challenge, the Horizon project proposes to use the paradigm of Multi-Agent Systems (MAS) as a modeling foundation [5]. The multi-agent paradigm seems to be attractive to develop an automatic piloting system because of the properties of agents itself. These properties include autonomy, proactivity, adaptability, cooperating, and mobility. In addition, multi-agents systems are decentralized in nature, which is required by large-scale networking environments. In this project, however, we prove the concept of such automatic piloting system by evaluating a specific problem. It becomes reasonable to experiment the proposed solution in a representative case before expanding it to the large-scale problem because of the network complexity and the multitude of parameters the system must deal with. We aim at showing that it is possible to extrapolate our solution to the whole network evaluating its performance in a simpler case.

There are several techniques that can be used to develop context-aware systems. This technical report describes some of these techniques and presents our choices to develop the piloting system. Chapter 2 describes a general layered architecture for context-aware systems. The following chapters present our choices based on the layers of this architecture. First, Chapter 3 focus on the context information that can be acquired from physical and virtual networks. After that, Chapter 4 describes several techniques used to represent knowledge. Finally, Chapter 5 defines the actions that can be taken to pilot the virtual networks. Concluding remarks and ongoing work are discussed in Chapter 6.

Chapter 2

General Architecture for Context-Aware Systems

Context-aware systems have three basic functionalities: sensing, thinking, and acting. We follow the general abstract layered architecture proposed by Baldauf *et al.*[6] shown in Figure 2.1. In this general architecture, the three functionalities are represented by subsystems. Each subsystem is composed of one or more layers associated to each other in order to exchange information. Each subsystem, however, may be decoupled or tightly integrated into one device, i.e., a subsystem can think and act but uses a shared set of sensors to acquire context information. In addition, the levels of complexity of each one of these functionalities are independent. We can develop, for example, a system that has complex sensors but performs little reasoning before taking actions. Furthermore, the three basic subsystems can be implemented in a centralized or in a distributed fashion.

Sensing is a context-aware subsystem that can be divided into two layers (Figure 2.1): sensors and raw data retrieval. The first layer consists of a set of sensors. We assume that sensors are every data source which provides context information no matter "what" is the data source. In this sense, a data source for temperature can be a hardware device, such as a thermometer, or a software module, such as a software application that requests the temperature to a Web service. Both are considered sensors because they provide temperature readings to the system. The second layer of the sensing subsystem is responsible for the retrieval of raw context data. This layer provides to the upper layers more abstract methods to request context information acquired by sensors. Actually, this layer is an interface to make implementation details of sensors transparent to the thinking subsystem. Hence, we can modify the sensors used by the system without modifications in the upper layers.

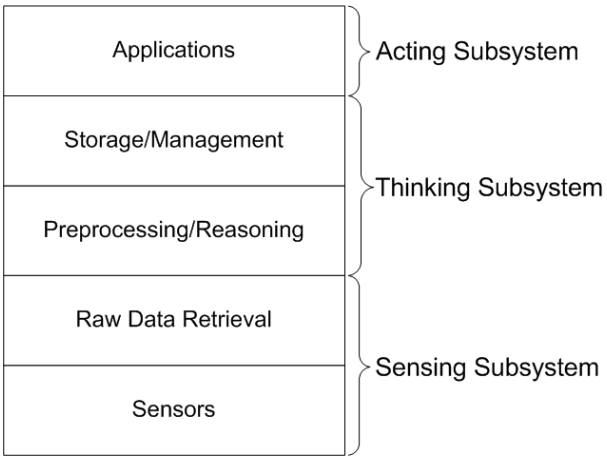


Figure 2.1: A general architecture for context-aware systems.

After collecting data with sensors, the task of the system is to use such data and to make sense of it [4]. This is the role of the thinking subsystem that can be divided in two layers: preprocessing and storage/management. The preprocessing layer converts all context information acquired into a common representation because the context information can be acquired in different forms, for example, such as discrete values or a continuous series of values. Furthermore, this layer is responsible for reasoning and interpreting context information to infer more knowledge. The reasoning technique used by the system ranges from simple event-condition rules to complex artificial intelligence techniques [4]. The preprocessing layer also aggregates context information from different sensors to provide more accurate information. In addition, knowledge representation techniques are needed to define and store context information in a machine processable form. This is the role of the store/management layer. There are several knowledge representation techniques, such as graphical-based, logical-based, and ontology-based techniques [6].

The third subsystem is acting. A context-aware system take actions based on the context information gathered or on situations recognized by the sensing and thinking subsystems. The actions to be taken by the system are defined according to the application requirements, as indicated in Figure 2.1. In a congestion control application, for example, sensors monitor the bandwidth in a network link and indicate that the link is saturated. Thus, the system "thinks" and decides to block new flows and to maintain all the current flows. In general, a context-aware system must act in time to quickly adapt its operation according to environment changes. In addition, these

systems allow users to control actions, i.e., users should be able to override, cancel, and stop actions and also reverse the results of an action.

In this project, we follow the described architecture to develop the piloting system. Our choice is to implement the layers in a distributed fashion based on multiagent systems. The sensing subsystem is implemented by sensors spread along the virtual routers. Thus, we do not have a sensor per agent but a set of sensors that can be used by all the agents. Sensors collect context information by reading directly data available on the operational systems of the routers and also by using measurement tools. We have also defined that sensors translate the context information acquired to a common representation. The context information considered in the project and the sensors under development are detailed in Chapter 3. The agents implement all the other layers related to thinking and acting subsystems. Thus, they must communicate with the set of sensors in order to acquire context information and, thus, process this information to take actions. The platforms for developing agents were described in Technical Report WP1-1.1 and in this technical report we present knowledge representation techniques in Chapter 4 and also the actions defined by the piloting system in Chapter 5.

Chapter 3

Sensing Subsystem

In this project, our choice is to develop a set of sensors decoupled of the agents. Thus, we are able to develop lightweight agents in terms of computational processing because they do not have to implement a sensor per context information to be sensed. The agents only have to exchange messages with sensors to request and to receive the context information desired. Therefore, although we slightly modify the classical architecture of agents by decoupling the sensing subsystem, every agent is still capable of acquiring sensing information.

The sensors that we are developing basically acquire context information by using two methods. Sensors read available data provided by operational systems running on physical and virtual routers and also use well-known tools for monitoring networks, such as `ping` [7], `nmap` [8], and `ifconfig` [9]. After collecting data, sensors translate the raw data into XML (Extended Markup Language) [10] data structures that are ready to be sent to agents when requested. In this technical report, we focus on the description of the context information that is useful to control the resource allocation among virtual networks. The architecture and the implementation of the sensors under development is detailed in Technical Reports WP2-2.2 and WP2-2.3.

In order to virtualize a network, we have to first define what resources we are planning to share among the multiple virtual networks. Currently, we have identified the following basic resources: processing power, memory, bandwidth, traffic, and network topology [11]. The computational resources of the network elements, such as routers and switches, must be sliced among the virtual networks. These resources include processing power and memory. It is worth mentioning that the resource sharing is a fundamental requirement to make the system work properly. A router can stop to forward packets and to exchange routing messages if it has not CPU cycle available. Memory isolation is an important requirement to avoid overriding of one routing table

on another virtual router when a routing table of a given virtual router increases and, thus, packets will be not correctly forwarded. Bandwidth, traffic, and topology are the network resources that we have to slice. First, we have to guarantee to each virtual network its own fraction of bandwidth on a physical link. All networks elements on a given source-destination path that are able to limit the forwarding rate should also be sliced [11]. Second, we have to be able to associate a specific set of traffic to one or more virtual networks, i.e., a set of traffic must be isolated from another. This is a key point to employ multiple virtual networks. In this sense, traffic means all packets to/from a given address or all HTTP traffic, for example. Finally, each network element should be aware of the nodes within a given virtual network. Thus, a virtual router should be able to determine its own view of other routers and the connectivity among them.

Currently, we are developing sensors to acquire context information related to the basic resources. Thus, the data that we can acquire from physical and virtual networks are divided in two classes: computational resources and network states. The first one is related to each physical and virtual router. On the other hand, the second one indicates the states of each physical and virtual link.

For each physical router, our sensors are currently able to acquire the following data depending on the network virtualization technique employed:

- Processor usage (physical CPUs)
- Used memory
- Available memory,
- Used swap memory
- Available swap memory
- Total memory allocated to a given virtual router
- Number of virtual routers per physical router
- Number of virtual processors allocated to a given virtual router and
- Number of virtual interfaces defined to a given virtual router

In this case, sensors directly read most of the desired data from the operational system of physical routers. We estimate physical processor usage by summing the usage of virtual processors.

For each virtual router, the following data are acquired by sensors:

- Processor usage (virtual CPUs)
- Used memory
- Available memory
- Used swap memory
- Available swap memory

According to the employed virtualization technique, the virtual CPU usage is obtained by using tools provided by the operational system of physical routers and the memory information are directly read from the operational system of virtual routers. In this case, we need to access these virtual routers.

The second class of context information is related to network states. Our sensors are currently able to acquire the following information per physical or virtual network interface:

- Number of received packets
- Received bytes
- Number of erroneous packets
- Number of dropped packets during receptions,
- Receiving rate
- Number of transmitted packets
- Transmitted bytes
- Number of erroneous transmitted packets
- Number of dropped packets during transmissions
- Transmission rate.

Sensors also discover the topologies of physical and virtual networks. Thus, for each neighbor, we can determine, for example, the available bandwidth and the latency of a given link. In our project, we consider that the context information previously presented is enough to serve as a basis to the reasoning techniques employed by the piloting system under development. With this information, the piloting system will be able to react to environment changes in order to guarantee the isolation and the resources of each virtual network.

Chapter 4

Thinking Subsystem

The role of the thinking subsystem is to make sense of data acquired by sensors and to use context information to react to environment changes. Basically, thinking subsystems combine knowledge representation and reasoning techniques. Knowledge representation techniques define and store context information in a machine processable form [12]. Thus, the goal of a knowledge representation technique is to store the context information in a logical form in order to allow that reasoning techniques use this information. The reasoning techniques include, for example, mathematical models, inference techniques, and cognitive-based models [4]. In this technical report, we are focusing on the knowledge representation techniques. The techniques used to reason with context information will be presented and discussed in the Technical Report WP3-3.2.

Several techniques are proposed to represent knowledge and there are studies that try to classify these techniques in classes [4, 13, 6]. Four of the most relevant classes of knowledge representation techniques are briefly described in the following paragraphs.

Markup-based techniques define hierarchical data structures to represent context information. These data structures are composed of markup tags with attributes and content. The content of each tag might be recursively defined by other tags. One of the most popular markup languages is XML [10]. Currently, there are several XML-based knowledge representation languages and standards, such as DARPA Agent Markup Language (DAML) [14] and Web Ontology Language (OWL) [15]. We are currently using XML as the common language to represent all data acquired by sensors and also to describe the content of the exchanged messages in the system.

In general, a logic derives a concluding expression from a set of expressions or facts based on predefined conditions [13]. This process is called inference and the conditions are formally described by a set of rules. Consequently,

context information is represented by facts, expressions, and rules if we are considering logic-based techniques. Thus, context information is added to a logic-based system in terms of facts as well as it is deleted or updated. Context information may be also inferred from the rules defined in the system. Ranganathan and Campbell [16], for example, propose to represent context and situations by using first-order logic techniques. In this case, rules are defined to map situations to actions by using Prolog language. The proposed rules basically relate context information to situations. Context information is the condition of a rule and situations are the conclusions of a rule.

Graphical modeling techniques are largely used because of its intuitive nature. The Unified Modeling Language (UML) [17] is a well-known general-purpose modeling tool and can also be used to represent context information [18]. The UML class diagrams are the graphical components of this language and, from these diagrams, we can derive entity-relationship models [19]. This kind of model is largely used as structuring tool for developing relational databases, which can be viewed as a knowledge base [13]. Another graphical technique is called contextual graphs [20, 21]. This technique does not define diagrams, like UML defines, but it proposes the concept of context spaces, which is a spatial view of context information. Each type of context information is represented by one axis of a multidimensional space and, thus, sensors readings are represented by points and situations are represented by regions in this space.

An ontology is generally defined as a set of concepts and terms that are used to describe a domain of knowledge or to develop a representation of this domain, including the relationships between their elements. Particularly, the set of terms can be ordered in a hierarchical fashion and used as a "sketch" to build a knowledge base [22]. From this definition, we clearly identify the differences between ontologies and knowledge bases. An ontology provides a set of terms to describe a given domain while a knowledge base uses these terms to describe a given situation. If this situation changes, the knowledge base also changes. The ontology, however, does not change because the domain remains the same. Thus, we can easily develop a knowledge base from an ontology, which is the main advantage of this technique. Furthermore, the ontology-based techniques have other advantages. First, these techniques avoid uncertainty because they provide an exact description and a specific vocabulary to represent knowledge. In addition, ontology-based techniques allow knowledge sharing because applications within the same domain can use the same ontology. Finally, the same ontology can be expressed in different languages. In general, an ontology is composed of: a taxonomy, i.e., a set of concepts and a hierarchy between them; a set of relationships between these concepts; and a set of axioms [22]. In Horizon project we have to

build a knowledge base for the piloting system, thus, our choice is to define an ontology to describe the environment with multiple virtual networks. In addition, most of the analyzed platforms for developing agents have tools to define ontologies inside their agents. The Ginkgo Platform [23, 1] considers an ontology-based representation composed of *classes* and *individuals*. Both concepts are quite similar to the classes and instances of an object data-model. An individual is an instance of a class. Hence, a class is a set of individuals, which are members of this class, and the knowledge base is a tree of classes, as illustrated by Figure 4.1. In this example, we have two classes, *person* and *employee*, that derives from the root class, *thing*, and one instance of each class, *John* and *Jane*. In practice, classes can be routers, users, flows, applications, etc. Furthermore, we can observe that classes have properties to store data. These properties have an identifier and are single or multivalued. The individuals have the same properties of the class they are members and also the properties of their father classes.

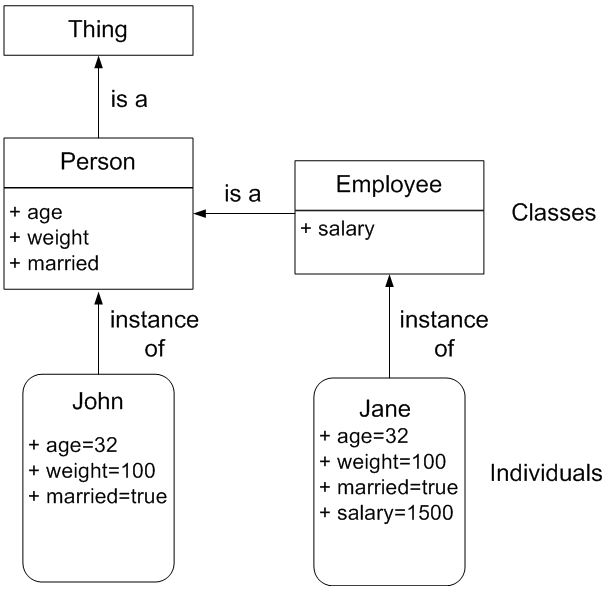


Figure 4.1: A simple knowledge base using the Ginkgo Platform [1].

Chapter 5

Acting Subsystem

The acting subsystem is the part of the piloting system responsible for reacting to environment changes based on the context information gathered or on situations recognized by the sensing and thinking subsystems. In this project, our choice is to define typical basic functionalities regardless the virtualization technique employed by the network. The actions taken by the piloting system are then based on these functionalities, which are following described.

We identified four basic functionalities: creation of multiple customized networks, flexible management, real-time control, and monitoring. We also defined primitives in order to employ these functionalities. The primitives allow to *instantiate/delete* and also to *migrate* network elements and flows and *set* its resource-allocation parameters. Such primitives make the network virtualization a suitable technology for creating multiple virtual networks and, as a consequence, for supporting the pluralist approach, because several requirements are satisfied, as explained below.

The first functionality is the creation of multiple customized networks. In a pluralist architecture, we have multiple networks running in parallel. In this sense, the instantiate primitive can be used to instantiate virtual network elements, such as virtual routers and/or virtual links, and, therefore, multiple virtual networks can be rapidly deployed and run simultaneously. Each virtual network has its own protocol stack, network topology, administration policy, etc. This enables network innovation and new business models [24]. With network virtualization, a service provider can allocate an end-to-end virtual path and instantiate a virtual network tailored to the offered network service, e.g. a network with quality-of-service (QoS) guarantees. Hence, new services can be easily deployed and new players can break the barrier to enter in the network service market.

Flexible Management is the second functionality that we have identified.

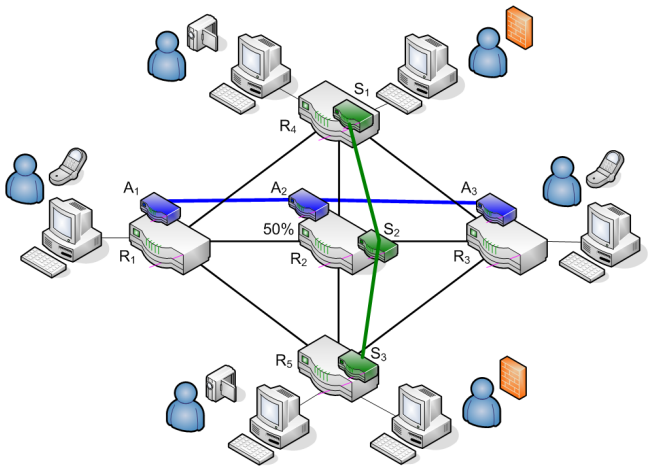
The network virtualization layer breaks the coupling between the logic used to construct the forwarding table and the physical hardware that implements the packet forwarding task [25]. Therefore, the migrate primitive allows moving a virtual network element from a physical hardware to another, without changing the logical/virtual network topology. In addition, traffic engineering and optimization techniques can use the migrate primitive to move virtual network elements/links along the physical infrastructure in order to minimize energy costs, distance from servers to specific network users, or other objective functions.

Real-time control is the third functionality. The virtual networks architecture also supports real-time control of virtual network resources because resource-allocation parameters can be set for each virtual network element (router, switch, link, gateway, etc.). We can set the allocated memory, bandwidth, maximum tolerated delay, etc. Even specific hardware parameters can be set, for instance, the number of virtual processors, priority of processor usage in a contention scenario, etc. Thus, we can dynamically adapt the resources allocated to each virtual network according to the current network condition, number of users, priority of each virtual network, service level agreements (SLAs), etc.

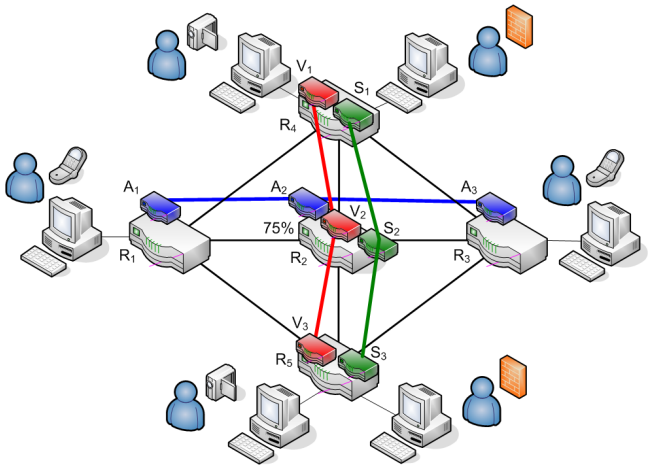
The last functionality is monitoring. Network virtualization techniques should come with a set of monitoring tools required to measure variables of interest, such as available bandwidth, processor and memory usage, link and end-to-end delay, etc. The monitor primitive is called to measure the desired variables. In this project, the measurements are performed by sensors as explained in Chapter 3. Thus, when the monitor primitive is called actually the sensors are called.

The four functionalities can be used by the piloting system to guarantee the requirements of each virtual network. For example, a given virtual network employs an intrusion detection system (IDS) in order to detect malicious nodes. In this case, the delete primitive can be used to delete a virtual network element/link or even an entire network if an attack (e.g., Distributed Denial of Service - DDoS) is detected. Figure 5.1 also shows an example of traffic engineering that uses the migrate primitive to move virtual routers along the physical infrastructure to minimize energy costs or other objective functions. In this example, we have five physical routers (R_1, R_2, R_3, R_4, and R_5) and initially two virtual networks. The first virtual network provides quality of service (QoS) for voice-over-IP (VoIP) calls and is composed of virtual routers A_1, A_2, and A_3 placed, respectively, in physical routers R_1, R_2, and R_3. The second one is a secure network composed of virtual routers S_1, S_2, and S_3 located, respectively, in physical routers R_4, R_2, and R_5. Assume that we have a rule to move the virtual router with the highest

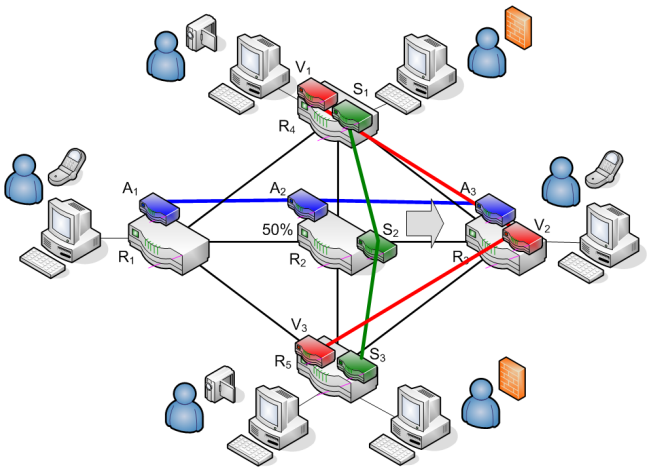
CPU usage just after the physical CPU usage reaches 80%. In this scenario, illustrated by Figure 5.1(a), the physical router R_2 has two virtual routers A_2 and S_2 and each one uses 25% of the physical CPU. Thus, the rule defined is satisfied. Suppose now that a third virtual network is created to provide QoS for video streaming applications. As illustrated by Figure 5.1(b), this networks is composed of virtual routers V_1, V_2, and V_3 located, respectively, in physical routers R_4, R_2, and R_5. Thus, we add one virtual router to R_2 and V_2 requires more 25% of the physical CPU. Currently, the CPU usage is 75% and the rule is still satisfied. After that, however, the virtual router V_2 requires more 10% of CPU and the total CPU usage becomes greater than 80%. Thus, according to the rule defined, we have to move the virtual with the highest CPU usage, which is V_2 in this case. An alternative is to move V_2 to R_3, as shown Figure 5.1(c). After migrating the virtual router, the CPU usage in all physical routers is less than 80% and the rule satisfied.



(a) Two virtual networks and 50% of CPU usage in R_2.



(b) Three virtual networks and 75% of CPU usage in R_2.



(c) Virtual router V_2 migrates to physical router R_3.

Figure 5.1: Traffic engineering using the migrate primitive.

Chapter 6

Conclusions and Ongoing Work

Horizon project is developing the infrastructure required for a piloting system to control the resources allocated to each virtual network. This system creates and destroys virtual networks, sets their parameters, and migrates network elements. These actions are taken based on the context information acquired by sensors spread along network elements. The piloting system is based on the multi-agent paradigm, developed in a distributed fashion to increase network scalability.

The piloting system also follows the layered architecture presented in Chapter 2. The sensors under development read available data provided by operational systems running on physical and virtual routers and also use well-known tools for monitoring networks. Sensors, however, are decoupled from agents to make them lightweight in terms of computational processing. In this case, agents only have to exchange messages with sensors to request and to receive the context information desired. Currently, sensors do not send messages to agents. After collecting data, sensors translate the raw data into XML data structures that are sent to a server, called Virtual Machine Server that will be described in Technical Report WP2-2.2. The next step is to evaluate the communication between sensors and agents.

The reasoning and knowledge representation techniques are under specification as well as the platform for developing agents. We are now evaluating Ginkgo and JADE platforms and both allow us to define an ontology and to build a knowledge base from this ontology. This is the reason to consider ontology-based techniques our preliminary choice to represent knowledge. Currently, all the reasoning is made by users that have access to the functions provided by the Virtual Machine Server. After the definition of the agent platform and the reasoning techniques, we will start to migrate these functions to the agents.

The actions defined for the piloting system are: creation of multiple cus-

tomized networks, flexible management, real-time control, and monitoring. In addition, the primitives defined to employ these functionalities are: instantiate, delete, and migrate network elements and flows and set the resource-allocation parameters. These functionalities and primitives are feasible regardless the virtualization technique employed by the network. The next step is to analyze if these functionalities and the network resources defined are enough to pilot the network.

Bibliography

[1] Ginkgo Networks, "Ginkgo agent platform programming manual v1.5," tech. rep., Ginkgo Networks, Oct. 2009.

[2] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *IEEE Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.

[3] J. He, R. Zhang-Shen, Y. Li, C.-Y. Lee, J. Rexford, and M. Chiang, "DaVinci: dynamically adaptive virtual networks for a customized Internet," in *ACM CoNEXT Conference*, Dec. 2008.

[4] S. Loke, *Context-Aware Pervasive Systems: Architectures for a New Breed of Applications*. Auerbach Publications, 1 ed., 2006.

[5] V. Silva and C. J. F. Lucena, "Modeling multi-agent system," *Communications of the ACM*, vol. 50, no. 5, pp. 103–108, May 2007.

[6] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, June 2007.

[7] Linux.org, *Ping Man page*, June 2010. Available at http://linux.die.net/man/8/ping.

[8] Nmap.org, *Nmap Reference Guide*, June 2010. Available at http://nmap.org/book/man.html.

[9] F. N. van Kempen, A. Cox, P. Blundell, and A. Kleen, *Ifconfig Man page*, June 2010. Available at http://linux.die.net/man/8/ifconfig.

[10] L. Quin, *Extensible Markup Language (XML)*, June 2010. Available at http://www.w3.org/XML/.

[11] R. Sherwood, G. Gibby, K.-K. Yapy, G. Appenzellery, M. Casado, N. McKeown, and G. Parulkar, "FlowVisor: A network virtualization layer," tech. rep., Deutsche Telekom Inc. R&D Lab, Stanford University, and Nicira Networks, Oct. 2009.

[12] R. Davis, H. Shrobe, and P. Szolovits, "What is a knowledge representation?," *AI Magazine*, vol. 14, no. 1, no. 1, pp. 17–33, 1993.

[13] T. Strang and C. Linnhoff-Popien, "A context modeling survey," in *International Workshop on Advanced Context Modelling, Reasoning and Management*, 2004.

[14] DARPA team, *The DARPA Agent Markup Language Homepage*, June 2010. Available at http://www.daml.org.

[15] D. L. McGuinness and F. van Harmelen, *OWL Web Ontology Language Overview*, June 2010. Available at http://www.w3.org/TR/owl-features/.

[16] A. Ranganathan and R. H. Campbell, "An infrastructure for context-awareness based on first order logic," *Personal and Ubiquitous Computing Journal*, vol. 7, no. 6, pp. 353–364, Dec. 2003.

[17] Object Management Group, *UML Resource Page*, June 2010. Available at http://www.uml.org/.

[18] P. Kogut, S. Cranefield, L. Hart, M. Dutra, K. Baclawski, M. Kokar, and J. Smith, "UML for ontology development," *The Knowledge Engineering Review*, vol. 17, no. 1, no. 1, pp. 61–64, 2002.

[19] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML class diagrams," *Artificial Intelligence*, vol. 168, no. 1-2, no. 1-2, pp. 70–118, 2005.

[20] P. Brézillon, "Representation of procedures and practices in contextual graphs," *The Knowledge Engineering Review*, vol. 18, no. 2, pp. 147–174, June 2003.

[21] A. Padovitz, S. W. Loke, and A. B. Zaslavsky, "Towards a theory of context spaces," in *Workshop on Context Modelling and Reasoning (COMOREA)*, pp. 38–42, 2004.

[22] A. Gómez-Pérez, "Ontological engineering: a state of the art," *Expert Update*, vol. 2, no. 3, no. 3, pp. 33–43, 1999.

[23] Ginkgo Networks, "Ginkgo distributed network piloting system," tech. rep., Ginkgo Networks, Sept. 2008.

[24] N. Feamster, L. Gao, and J. Rexford, "How to lease the Internet in your spare time," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 1, pp. 61–64, Jan. 2007.

[25] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, no. 4, pp. 231–242, 2008.