Mobile Netw Appl (2015) 20:19–31
DOI 10.1007/s11036-014-0536-5

Multi-perspective virtualization and software-defined infrastructure framework for wireless access networks

Heming Wen · Prabhat Kumar Tiwary · Tho Le-Ngoc

Published online: 11 October 2014

Abstract Virtualization and service-oriented architecture are important concepts that triggered the rapid evolution of cloud computing and software-defined technologies. Since wireless technologies will play an important role in the future of networking technologies, this article presents Aurora, a virtualization framework and testbed platform for supporting multiple types of virtualization techniques and architectures specifically applied to wireless technologies. Firstly, a brief background overview of the three main perspectives of wireless virtualization based on the type of virtualized resources and the depth of slicing is provided along with some of the challenges and requirements for a sustainable and generic wireless virtualization framework. Secondly, the software architecture and the design principles behind Aurora are explained. Aurora is designed to fulfil multiple roles as a powerful tool to combine multiple wireless virtualization technologies, a software-defined research platform for developing new virtualization architectures and a service-oriented wireless infrastructure manager. Lastly, an instance of the software implementation of Aurora is presented in order to demonstrate the feasibility of deployment of the framework.

Keywords Wireless virtualization · Software-defined networking · Software-defined radio · Service-oriented architecture · Future internet · Cloud infrastructure

H. Wen (✉) · P. K. Tiwary · T. Le-Ngoc
Electrical and Computer Engineering, McGill University, 3480 rue University, Montreal, Canada
e-mail: heming.wen@mail.mcgill.ca

P. K. Tiwary
e-mail: prabhat.tiwary@mail.mcgill.ca

T. Le-Ngoc
e-mail: tho.le-ngoc@mcgill.ca

1 Introduction

Recently, virtualization has gained important momentum in the fields of computing and networking. Server virtualization aided by network virtualization has led to a service-oriented infrastructure harnessing technologies such as software-defined networking (SDN) e.g., OpenFlow [1] and cloud management and orchestration system, e.g., OpenStack [2]. These trends lead to the development of a new system architecture and business model that focuses on offering virtualized hardware resources as a service. As for wireless virtualization, the two main driving forces acting behind it are: incessant emergence of new wireless services and applications, and research in software-defined technologies and service-oriented paradigm. For instance, new radio technologies and wireless transmission techniques, ranging from software-defined radio (SDR), such as OpenRadio [3], and cognitive radio to coordinated multi-point (CoMP), are aimed at making the wireless communication infrastructure more dynamic and efficient. Advancements in optical fibre technologies make architectures such as the fibre-connected massively-distributed antennas (FMDA) system feasible [4]. All these technical advancements are like pieces of a larger puzzle that have the potential to generate a "perfect storm" which could result in a major revolution of the wireless landscape: the transition and convergence of the wireless infrastructure into a *virtualized*, *software-defined* and *service-oriented* infrastructure.

Wireless virtualization and software-defined infrastructure (SDI) have a wide range of potential applications. From a commercial perspective, virtualization can lower the capital expenditures for the deployment of new wireless services by enabling a flexible sharing of the existing infrastructure [5]. Within the scope of a wide-scale deployment, the decoupling of the infrastructure from its functionalities and consolidation of the management of these functionalities into the cloud are ways to allow the wireless infrastructure to be provided as a

service instead of an asset. As such, wireless virtualization can benefit the application of service-oriented architecture on the wireless infrastructure. Virtualization can be used to enable multiple network operators and service providers to offer differentiated services over the same infrastructure, as suggested in [6]. Furthermore, virtualization is fundamental to research themes such as Future Internet [7] and clean-slate design [8]. Virtualization can shorten the research and development life cycle of new wireless technologies by providing a more open and flexible infrastructure [9].

There are many approaches to wireless virtualization since different levels of virtualization are required for different services and applications. They range from the infrastructure and network-oriented approaches, such as [6] and [10], to the more hardware-focused approach such as [3]. Our previous paper [11] contains an extensive survey of recent works in the field of wireless virtualization and a hypothetical ecosystem scenario of a future virtualized wireless infrastructure. This article, however, is focused towards defining a generic and modular wireless virtualization framework that enables the coexistence of multiple virtualization perspectives integrated across different wireless technologies, as well as a platform to experiment with these technologies However, virtualization by itself is only considered by this article as one of the enabling technologies. The application of virtualization becomes truly interesting when integrated with other modern principles. As such, this article expands beyond the scope of just resource virtualization and focuses on combining three different but related concepts, all applied within the context of the wireless infrastructure: *virtualization*, *software-defined principles* and *service-oriented architecture*.

The rest of the article is organized as follows. Section 2 discusses the concept of wireless virtualization through the identification of different virtualization perspectives. Section 3 presents challenges and defining requirements of a generic wireless virtualization framework. It also outlines a wireless virtualization framework that can satisfy these requirements. In Section 4, the component architecture of a virtualization and software-defined infrastructure software platform for wireless access networks, codenamed *Aurora*, is presented. Section 5 discusses an example implementation of Aurora in the existing OpenStack cloud infrastructure platform and the Smart Applications on Virtual Infrastructure (SAVI) testbed [12]. Finally, Section 6 concludes this article, highlights the potential module extensions to Aurora and outlines the future research directions.

2 A brief overview of wireless virtualization

Many active research projects on wireless virtualization and related enabling technologies exist. They can be grouped by scope (e.g., general architecture vs. specific implementations) and technologies (e.g., WiFi vs. cellular). Readers are encouraged to refer [11] and [13] for a more detailed and exhaustive survey of wireless virtualization as well as network virtualization in the context of wireless access nodes. A classification of wireless virtualization perspectives is vital to the design of a generic and modular wireless virtualization framework. Based on the characteristics of the resources being virtualized, the different approaches to wireless virtualization can be classified into three different perspectives.

(1). *Flow-based Wireless Virtualization*: In flow-based wireless virtualization, the focus is on the customization and control over the datapath of the wireless infrastructure, which consist of the isolation, scheduling, management and service differentiation among traffic flows. Flow-based virtualization is the most feasible approach to implement which leads to a more flexible and efficient traffic and resource management. In addition, it can be used for the integration of wireless technologies with the rest of the cloud infrastructure.

(2). *Protocol-based Wireless Virtualization*: Protocol-based wireless virtualization focuses on the isolation, customization and management of multiple wireless protocol instances on the same radio hardware. As opposed to the flow-based virtualization where tenants share the same wireless protocol stack, this perspective allows tenants to control separate instances of the wireless protocol stacks on the same radio hardware. As such, the types of resources being virtualized are mostly MAC and PHY processing resources.

(3). *Spectrum and RF Frontend Virtualization*: The spectrum and RF frontend virtualization, the most granular perspective of virtualization, focuses on the dynamic allocation and management of the spectrum and the radio frontend nodes. This perspective aims at providing an abstraction layer over the spectrum available at a given region and a given time in order to support a more intelligent, flexible and efficient spectrum usage. It provides a wireless standard-independent abstraction layer.

According to [14], cross-domain integration of control and customization allows for a more agile infrastructure. This integration allows the infrastructure to be an active part of the application or the service, not simply a support. For instance, wireless virtualization can be integrated with other domains in order to form a *heterogeneous cloud infrastructure* in which a single slice can span across the entire infrastructure, binding together virtualized computing, networking and wireless resources. The different wireless virtualization perspectives are complementary to each other and can play different roles in a heterogeneous cloud infrastructure. Furthermore, not all the perspectives can reach their full potential with currently available technologies and techniques. With the evolution in

enabling technologies, it becomes possible to graft new virtualization perspective in the infrastructure which was not supported before. This is particularly true for protocol-based virtualization, which requires a mature and sustainable SDR platform, and spectrum virtualization, which requires new radio hardware architectures. Thus, the availability of technology and the transition from one technology to another enforce the idea of coexistence among different perspectives.

3 Towards a generic wireless virtualization framework

3.1 Challenges of a generic virtualization framework

A wireless virtualization framework is truly generic if it provides flexibility in three different aspects: (i) support for different virtualization perspectives, (ii) ability to integrate existing and future enabling technologies for virtualization, and (iii) capacity to provide diverse virtualized infrastructure. Some of the challenges of wireless virtualization framework design have been identified in [11] and summarized as follows.

(1). *Flexibility-performance trade-off and scalability*: It all comes down to the question of determining the 'right amount' of wireless virtualization between flexibility/abstraction and performance. Unfortunately, since different applications and services have different requirements, there is no right answer to this question. Thus, a virtualization framework should delegate as much as possible the direct management of resources to tenant-owned controllers/managers, as implied in [9], giving each tenant the flexibility to build its own virtual infrastructure while retaining the performance of the technologies being used.

(2). *Complexity of the framework*: By design, a virtualization layer (or *hypervisor*) performs virtualization-related functionalities such as slice isolation, function translation, policy reinforcement and multiplexing. A good way to avoid increasing the overall complexity is to abstract and modularize these operations into different API layers. Since tenants have differing needs and understanding of the underlying technologies, the framework should aim at being user-friendly without removing access to the complexity actually required by certain tenants.

(3). *Feasibility of deployment*: The implementation of any framework requires technology-specific integration. However, unlike server virtualization which is mainly dominated by a single technology (×86 architecture), wireless technologies are diverse. Therefore, a multi-perspective framework approach should be aimed by designing a modular platform on which different virtualization technologies can be gradually integrated. Moreover, the framework has to be backward compatible and retrofitting.

3.2 Defining requirements for a generic virtualization framework

The different approaches to handle the challenges can be formalized as requirements and features of a generic multi-perspective virtualization framework. *Aurora* is designed based on the following requirements:

(1). *Generic, modular and open framework*: In order to support a heterogeneous infrastructure with different perspectives, there are many characteristics that need to be considered. Firstly, the *core framework* should be technology-independent and act as an engine to orchestrate and broker wireless resources from an abstracted point of view. This allows the framework to be easily reconfigurable and extensible. Secondly, *modularity* is an important characteristic that should be applied to both internal (between modules) and external (exposed to tenants) interfaces and functions. Technology-specific functions should be self-contained in a replaceable software module. Lastly, the framework itself should be *openly accessible* by the tenants such that it allows them to setup and configure their own data and control path and implement their own virtualization and software-defined architecture inside the framework.

(2). *Evolvability and extensibility of the framework*: Since it is not required to have all three wireless virtualization perspectives integrated at the same time, a progressive approach adjusted to different applications is suggested. As the depth of virtualization evolves, new extension modules and plug-ins can be added to the framework. The potential integration and federation of both existing and emerging virtualization architectures should be considered inside the framework so as to avoid reinventing the wheel or becoming obsolete.

(3). *Resource and function abstraction*: In line with the modularity requirement, different levels of abstraction should be applied to the resources and functions of a virtualized infrastructure. The resources should be abstracted as building blocks, giving a sandbox-like view of the infrastructure. A tenant of the infrastructure can then provide custom virtualization services to other tenants, forming an interesting ecosystem of service exchange among tenants on the same infrastructure [11].

3.3 Outline of a generic virtualization framework

Based on these challenges and requirements, a basic outline of a *generic* wireless virtualization framework has been put forward as shown in Fig. 1. The framework has two main layers: the *control and management layer* (CML) and the *virtualization layer* (VL). This architecture largely extends

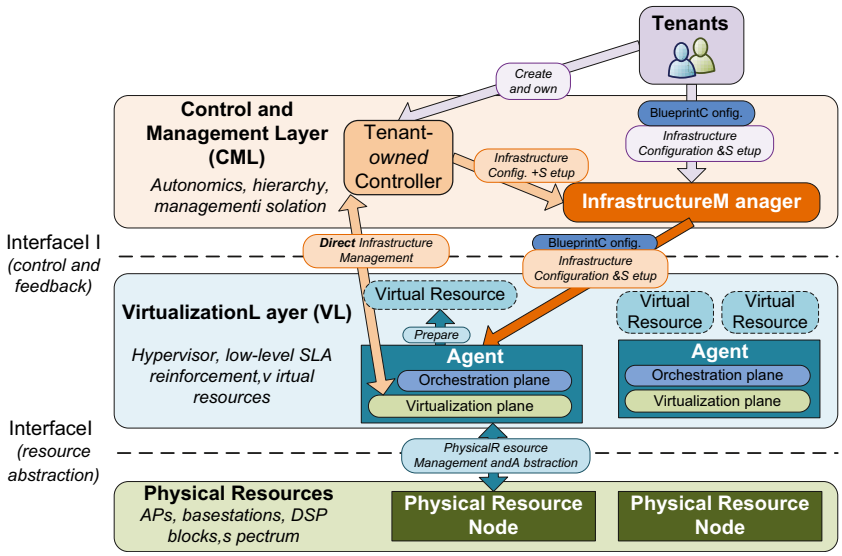

Fig. 1 A Basic Outline Framework for Wireless Virtualization

the Virtual Radio architecture in [9], which described resource management and brokerage through a virtualization manager interface (VMI). The modularity of the interfaces is inspired by the MAC abstraction layer for mesh networks in [15]. Compared to other frameworks, the proposed framework considers the gradual integration of other wireless virtualization technologies into itself and the evolution and emergence of new virtualization technologies. Comparison between the proposed framework and the existing works will be discussed in Section 4.3.

The CML contains a series of components that process wireless infrastructure-wide management functions. It encompasses both the infrastructure manager and the tenant-owned managers/controllers. The tenants can deploy their own custom management functionalities using the *tenant-owned controllers*, while leaving the job of isolation and conflict resolution among tenants to the *infrastructure manager*. The VL includes a set of virtualization agents that can be configured by the tenants through the CML. These agents in virtualization layer have two planes: technology-independent *orchestration* or *agent plane* and technology-dependent *virtualization plane*. The orchestration plane contains abstracted modules corresponding to core engines for slice creation, deletion and modification for different virtualization perspectives and *orchestrates* technologies in the virtualization plane.

The basic operation of the framework is also highlighted in Fig. 1. First, the tenant sends a *configuration blueprint* to the infrastructure manager to request a specific virtual wireless resource. Alternatively, the tenant can always deploy its own tenant-owned controller. The infrastructure manager validates the request and sends a *slice creation* command to the agent located on the physical resource node. The *virtualization agent* prepares a virtual slice and sets it up to be connected to the tenant virtual network (s). A direct control and data path between the slice and the tenant is therefore established. By allowing a tenant to *directly* interact with the virtualization plane once the setup is complete, the actual performance of the (virtualization) technologies being used in the virtualization plane remains unaffected. The framework is not designed to make existing technologies more efficient, but to provide a more flexible and evolutionary paradigm.

3.4 Multi-perspective virtualization using the framework

Different perspectives of virtualization can be realized to obtain heterogeneous virtualized infrastructure using this framework. The integration of different perspectives within the infrastructure is illustrated in Fig. 2 with a few example cases of resources with varying degrees of virtualization.

(a). First, a non-virtualizable wireless resource can be integrated within the infrastructure by the addition of an interface agent. The agent can establish the connectivity between the resource and the tenant network. Legacy resources can then be directly allocated without any sharing.

(b). With a resource node that only supports the flow-based virtualization, the capabilities that can be offered to tenants include datapath isolation and customization. The agent on such resources must setup virtual interfaces, virtual bridges, tunnelling endpoints, traffic shaping mechanisms and SDN-enabled technologies. However, tenants must share the same wireless protocol and the same radio parameters.

(c). If protocol-based virtualization is supported, the resource node still retains the flow-based capabilities but acquire additional features such as the ability to allocate unique *radio configuration profiles* for each tenant. Since a resource node with more capabilities can offer virtual resources with lower capabilities, default radio

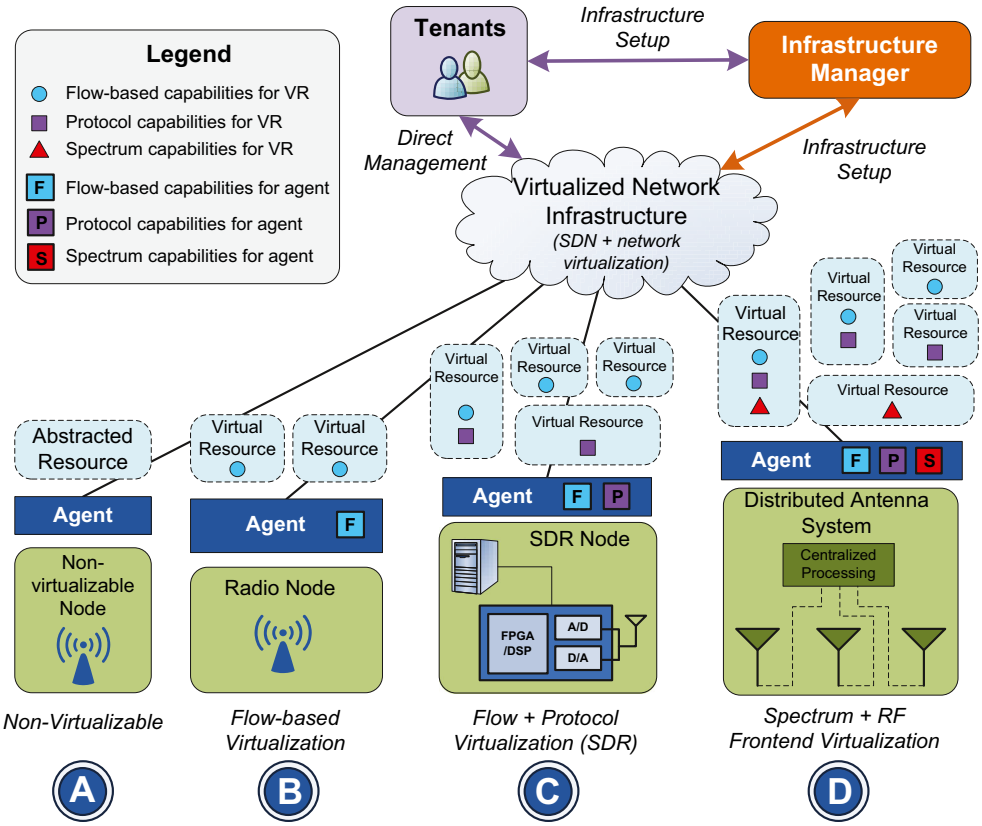

Fig. 2 Framework with Multi-Perspective Wireless Virtualization

configuration packages can be provided to tenants who only need flow-based virtual resources. This addresses some of the backward compatibility and retrofitting issue, as discussed in Section 3.1.

(d). In the advanced case where spectrum and RF frontend virtualization is supported, a network of radio nodes can be managed by a single agent, which can offer multiple types of virtual resources to different tenants. These resources can range from portions of a spectrum to a virtual radio node running a specific protocol.

4 Aurora virtualization platform architecture

As mentioned in Section 3.3, the general framework is greatly influenced by the architectures presented in [9] and [15]. This section presents more detailed implementation architecture in the form of *Aurora*. Aurora is designed to be closely related to the cloud infrastructure platform OpenStack [2] and the SAVI testbed [12]. OpenStack was chosen as a base framework due to its open-source nature and tight integration within the SAVI testbed. Aurora is a new service component that attempts to extend the OpenStack family of services by providing orchestration and virtualization tools for wireless resources with different wireless virtualization perspectives as identified in the Section 2. Aurora, by providing "OpenStack-like" control and management client console commands and representational state transfer (REST) APIs, makes wireless virtualization concepts easier to grasp for users who are familiar with OpenStack.

4.1 Resource abstraction model

Resource abstraction has been identified as one of the main requirements for a generic wireless virtualization framework in Section 3.2. In Aurora, there are three *classes* of resources abstracted in software: virtual wireless networks, virtual resource slices and physical resource nodes. The logical relationship between the different classes of resources is shown in Fig. 3. These abstracted resources are the building blocks of a virtual infrastructure in Aurora.

(1). Physical resource node (*wnode*): Different types of radio nodes are represented as different *flavors* of *wnode*, which include 802.11 access points (*ap*), software-defined radios (*sdr*), cellular base-stations (*base*) and other wireless technologies. Even within a type of resource node, the capabilities will differ depending on the implementation. In other words, two *ap* nodes do not necessarily offer the same level of services.

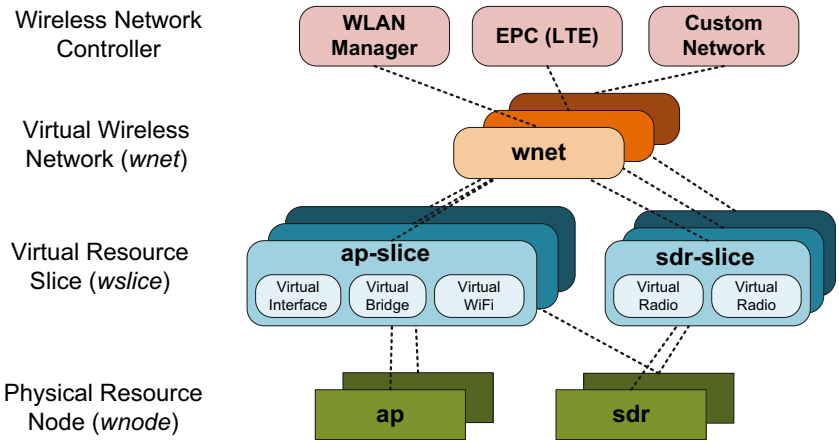

Fig. 3 Classes of Resources in Aurora

(a). *802.11 wireless access point* (*ap*): This flavor of *wnode* only supports flow-based virtualization with possible partial protocol virtualization. At the time of writing of this article, it only offers one type of virtual resource slice: *ap-slice*.

(b). *Software-defined radio* (*sdr*): The software-defined radio resource node can potentially support all three virtualization perspectives. It can offer more than one types of virtual resource slices. For instance, it can offer an instance of the SDR platform itself (*sdr-slice*) or an instance of a particular protocol standard, such as 802.11 (*ap-slice*). An example of SDR resources that can be integrated within Aurora is a radio node created using OpenRadio [3].

(2) *Virtual resource slice (wslice)*: The *wslice* is a virtual instance of the wireless resource node allocated to a tenant. It can be either a single virtual resource or a *package* and *container* of different virtual resources (bandwidth, VAP, etc.) which can be configured and controlled by the tenant. The *wslice* is defined by a setup blueprint specified by the tenants and validated by the Aurora infrastructure manager. Each *wslice* is owned by a single tenant and can join a *wnet*. Different flavors of wnode can support different varieties of *wslice*, such as *ap-slice* for 802.11 or *sdr-slice* for SDR.

(3). *Virtual wireless network (wnet)*: A *wnet* is a group of one or more *wslice*. It is a subnet-like abstraction of wslices with associated wireless network management functionalities. A *wnet* is connected to a wireless network manager, a software controller that manages the policies pertaining to virtual resources within that *wnet*. This manager can be a tenant-owned custom controller or a default controller provided by the Aurora framework.

4.2 Software components of aurora

The high-level Aurora architecture, as shown in Fig. 4, is divided into four main components. *Aurora-Client*, *Aurora-Manager* and *Aurora-Agent* are the core components of the framework whereas *Aurora-Tenant* is a set of additional tools for tenants that complement the framework.

(1) *Aurora-Client*: The Aurora-Client is the interface that provides the console commands for users to setup and manage their resources. It is designed to be similar to the OpenStack client. The Aurora-Client performs basic command validation and sends the commands as service requests to the Aurora-Manager.

(2). *Aurora-Tenant*: The Aurora-Tenant is a framework-provided "package" or VM image that allows tenants to deploy their own wireless network controller inside the virtual infrastructure. Even though it is not a mandatory component of Aurora, it aims at providing the facility to the tenants to deploy their own custom tools, which improve the accessibility of Aurora, similar to platform-as-a-service (PaaS) in cloud computing. These tools can range from SDN-based controllers to wireless network management software.

(3) *Aurora-Manager*: The Aurora-Manager is the infrastructure manager that performs three main roles: to provide access to infrastructure information, to redirect the setup and configuration of virtual wireless resources from the clients to the virtualization agents and to host wireless infrastructure resource allocation and management services. For the first two roles, a REST/HTTP API server handles both API requests from Aurora-Client instances from different tenants and API requests from Aurora-Tenants. Two persistent management databases are used to keep record of the status, attributes, capabilities and configuration of both physical and virtual resources. The *resource database* is a structured query language (SQL)-based database which contains only simple resource attributes (name, ID, tenant, status, etc.) and relationships between resources. Each class of resources has its own SQL table. The *configuration database* stores full *wslice* configuration blueprints for each tenant.

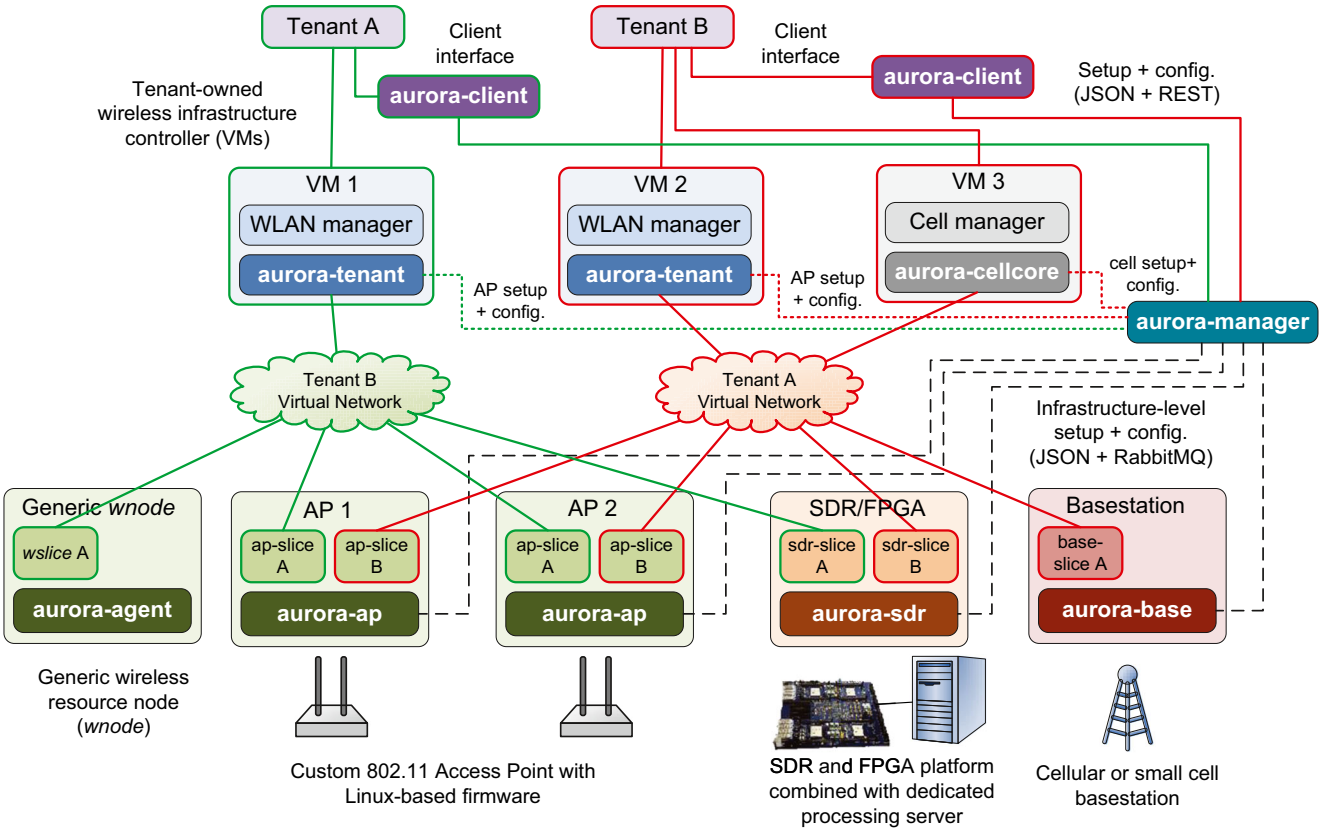

Fig. 4 Software Components of Aurora

For setup and configuration such as slice creation, the Aurora-Manager loads different *configuration modules* each in charge of specific software components of a *wslice*, such as *virtual interfaces*. The Aurora-Manager uses these modules to parse and validate the blueprint file obtained from the client. During this step, configuration conflicts such as invalid or duplicate names and unsupported capabilities are resolved. If the validation is successful, blueprint files are re-generated for each *wslice* and dispatched to the Aurora-Agent through *dispatch modules*. The final role of the Aurora-Manager is to host *management modules* that perform various wireless network management functionalities based on metrics obtained through *event modules*. These functionalities can include, but are not limited to, resource monitoring, metering services, mobility management services, dynamic provisioning of radio nodes, dynamic reconfiguration of the virtualized infrastructure [16], migration of virtual radios, outage handling and dynamic spectrum reuse. The various management heuristics that can be achieved through software is outside the scope of this paper. However, since the configurability of the Aurora nodes is flexible (supports multiple level of depth and layering) and modular (components can be configured independent from each other), various experimental algorithms, like the ones for network re-embedding described in [16], can be deployed to solve problems in virtual network management. The simplified architecture of the Aurora-Manager is shown in Fig. 5.

(4). *Aurora-Agent*: The Aurora-Agent is the local virtualization agent residing on the physical resource node and a key component in Aurora. It is to be noted that a specific implementation of an Aurora-Agent needs to be customized to work with a particular wireless technology, such as 802.11, SDRs and potentially cellular base-stations. The generic nature of the framework, however, comes from the design of the agent that involves a technology-independent *agent plane* and a technology-specific *virtualization plane*. The implementation covered in this article is *Aurora-AP*, a local virtualization agent for 802.11 access points. Aurora-AP's agent plane orchestrates the setup and configuration of other technologies, called *resource components* in the *virtualization plane*, through plug-ins grouped as *abstraction modules* in order to build the wireless slice (*ap-slice*). The actual slice isolation is implicitly achieved through resource components (virtualization plane) and not directly by the agent. As such, the operational performance of the virtualization is determined by the performance of the individual technologies and the complexity of the setup, not by the Aurora-Agent itself. The architecture of the Aurora-AP is shown in Fig. 6.

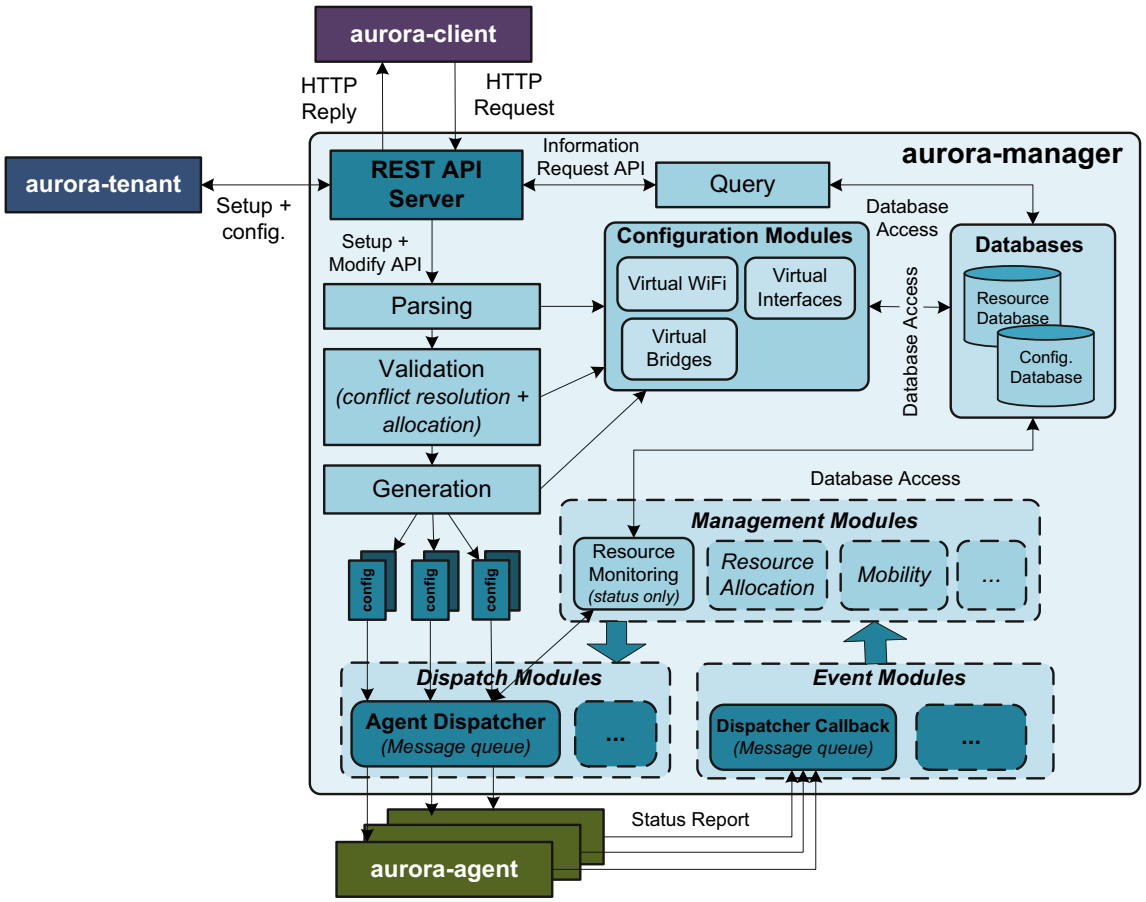

Fig. 5 Simplified Component Architecture of Aurora-Manager

In Aurora-Agent, abstraction modules are abstraction layers to specific technologies or resource components, interfaced through *plug-ins*. The architecture in Fig. 6 only has abstraction modules for basic flow-based virtualization. Aurora-Agent has a *local database* to store configuration information of wireless slices currently active on the resource node. The *core modules* contain the core logic of the agent and include the *agent initialization module* and the *ap-slice core* module. These modules execute various "routines" to perform various tasks requested by the Aurora-Manager or to handle special situations. These routines can be different for different virtualization perspectives. For example, the *agent initialization module* is called when the Aurora-Agent starts for the first time or when a node recovers from a failure. During this initialization, the local database synchronizes with the configuration database on the Aurora-Manager. The slices are then built/rebuilt on the physical node. The *AP-slice core module* handles all commands related to *ap-slice* instances. The basic routines implemented for this core module are: *ap-slice* creation, deletion, modification and restart.

The key point of Aurora-Agent is that all these modules, despite being responsible for defining the resources in Aurora, are not directly tied to the core engine of Aurora and can be exchanged with other technologies. After all, Aurora is a *meta*-framework that provides high level of customization flexibility and modularity to both the tenant users and the developers of the framework.

4.3 Aurora and related works

Wireless virtualization is still at its early stage of research and development. Instead of proposing a new virtualization technique or arguing for the adoption of one particular perspective, Aurora embraces the idea of coexistence between different virtualization perspectives targeted for different needs. This framework differentiates itself from other existing frameworks as it is both a flexible testbed platform and a powerful suite of tools similar to a wireless infrastructure "operating system" that enable full reconfigurability of the wireless infrastructure. More specifically, Aurora addresses the different challenges and requirements presented in Sections 3.1 and 3.2 by the following design considerations:

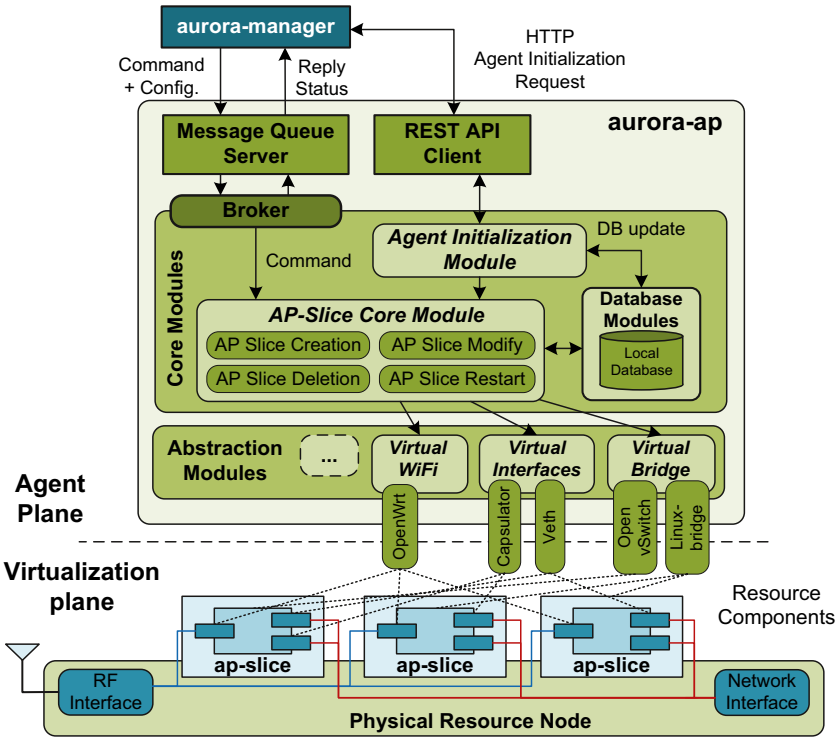

Fig. 6 Simplified Component Architecture of Aurora-Agent

(1). *Flexibility-performance trade-off and scalability*: One major selling point of this framework is that it adds no overhead to the performance of the virtualization technologies it integrates as it is not interfering with the direct data path of its component technologies. Aurora, as a framework, does not directly perform resource slicing and virtualization as it delegates the virtualization to other existing virtualization technologies. As such, it is as efficient (or inefficient) as the technology plug-ins that it uses for a given slice of the infrastructure. Most of the processing time of Aurora is spent during the setup and initialization of the infrastructure slice. Compared to the individual technologies, Aurora has the clear advantage that it can support different virtualization technologies and fulfill different needs with different solutions.

(2). *Complexity of the framework*: The additional layering provided by the Aurora plug-ins and abstraction modules are there to regulate and isolate the complexity of different technologies. Then, by borrowing the concept of "flavors" from cloud computing platforms, it can offer different levels of complexity to different tenants based on their needs.

(3). *Feasibility of deployment*: Aurora addresses this issue by borrowing elements from the existing SAVI infrastructure and the OpenStack platform in order to be deployed on a real infrastructure. Also, leveraging from its modular architecture, the deployment can also be more sustainable.

(4). *Generic, modular and open framework*: Aurora defines standard interfaces on how the resources are integrated within the framework using a modular orchestration architecture. For instance, different virtual interface implementations can be integrated within the same abstraction module, providing a library-like approach to the framework. In addition, Aurora can operate using purely open-source implementations and its architecture is transparent for further extensions. Tenants have the option of directly orchestrating their slices of resources or using existing functionalities of Aurora.

(5). *Evolvability and extensibility of the framework*: The way plug-ins are separated from the core modules of the framework through the abstraction layer allows anyone to develop their own plug-ins integrating virtualization technologies not yet available in Aurora without modifying the core components, facilitating the evolution of the framework.

(6). *Resource and function abstraction*: Aurora is highly based on resource and function abstraction at all layers of its architecture. As such, different resources with different capabilities can be managed through a generic core engine. For instance, Fig. 3 from Section 4.1 illustrates the relationship of the different resources that it manages.

Compared to existing works such as Virtual Radio [9] and CARMEN [15], the proposed Aurora framework borrows common concepts but also heavily expands on the scope of

the framework. For instance, Virtual Radio [9] provides the concept of a virtual radio node and an intermediate broker, which are incorporated into Aurora in the form of the Aurora-Agent and the wireless resource node (wnode). However, Virtual Radio by itself does not consider different virtualization perspectives and is much more limited in scope. It also lacks the perspective of integration with a cloud-based and service-oriented infrastructure like Aurora provides. As for CARMEN [15], the idea of technology-independent and dependent abstraction layers are translated into the abstraction modules and plug-ins in Aurora. However, Aurora considers the coexistence of multiple virtualization perspectives, not just types of wireless technologies in a heterogeneous network like CARMEN. The biggest advantage of Aurora over the existing framework is its tenant-configurable virtualization layer and strong integration with other virtualization domains, notably computer, network and infrastructure virtualization. Aurora provides a concrete example of how wireless virtualization can be related to existing cloud computing platforms like OpenStack and how it can be deployed in existing infrastructure like SAVI. Although similar to the NOS (network operating system) mentioned in [6], this paper emphasizes on the components that can be used as building blocks for such a system. For example, an analogy with operating system terminology would be that the NOS in [6] (and to some extent [10]) is similar to a program running inside the *virtualization operating system* Aurora, which is similar to a set of kernel modules.

5 Application and implementation of aurora

In fact, it would be difficult to analyze the performance of Aurora in its current incarnation because it is an orchestration framework that mainly attributes processing time during slice creation. The operational performance of a wireless infrastructure deploying Aurora will entirely depend on the setup blueprint and the component technologies used in the setup. In this section, some implementation examples and practical application scenarios are presented. Aurora represents a basic "virtualization operating system" implemented as a service platform that attempts to offer flexibility while providing useful tools for the development and implementation of wireless virtualization. Aurora is mainly implemented in Python [17]. This facilitates the integration of Aurora with OpenStack and SAVI testbed platform since Python is also the language of implementation for OpenStack and most of the SAVI testbed services. The relationships among the different resources are stored using a SQL database, more specifically MySQL [18]. The configuration blueprints are implemented as JavaScript Object Notation (JSON) files. Aurora-AP is implemented on PC Engines *alix3d2* APs [19] with two 802.11ab/g mini-PCI radio interface cards. The wireless firmware used for this particular instance of Aurora-AP is a customized OpenWrt [20] based on Linux kernel 3.2.

5.1 Aurora-AP: datapath and flow-based virtualization in aurora

One of the original challenges of a generic framework for wireless virtualization is to support a large variety of virtualization techniques and perspectives on the same infrastructure to satisfy different tenants. The solution provided by Aurora is that the tenant is given freedom to use the various resource components abstracted by Aurora-AP to build and *customize* their own *ap-slice*, which is very different from other frameworks.

In a typical enterprise-grade 802.11 AP, the concept of multiple virtual access points (VAPs) [21] assigns one basic service set (BSS) per virtual network. The general observation is that the data traffic must ultimately go through both the radio interface and the backbone network interface. Aurora-AP enhances the VAP by providing additional components to build and customize these traffic paths, in line with the concept of *flow-based* virtualization. Ultimately, the isolation of the traffic between the different *ap-slice* instances depends on each datapath component and the interaction between them. A "toolbox" approach is applied, in which the implementation of a particular function is selected from different *flavors* grouped inside an Aurora *module*. A specific *flavor* of a *module* is implemented as a *plug-in*. Three *modules* are used by Aurora-AP: virtual interfaces, virtual bridges and virtual WiFi radio. These resources are orchestrated to construct instances of *ap-slice*, as shown in Fig. 7. Each of these components is abstracted as a building block that can be declared in the blueprint configuration file.

(1). *Virtual (network) interfaces*: In Aurora-AP, different flavors of virtual network interfaces are provided to tenants, each fulfilling different roles. The *Capsulator* tunneling interface [22] is a custom over-IP tunneling program that provides basic flow isolation between tenants and is used in many research projects involving OpenFlow such as [23] and [24]. On the other hand, the Virtual Ethernet interface (*veth*) is a basic virtual interface that has its own MAC address and duplicates the traffic to and from another interface to which it is attached. In Aurora, these interfaces are attached to the virtual wireless interfaces to mask them as virtual network interfaces. They are also used to duplicate physical interfaces. Other types of interfaces such as virtual local area network (VLAN) can be developed separately as extension plug-ins to this module.

(2). *Virtual bridges*: In Aurora, the virtual bridges modules provide internal layer 2 connection between virtual interfaces. Virtual bridges bind two or more physical or

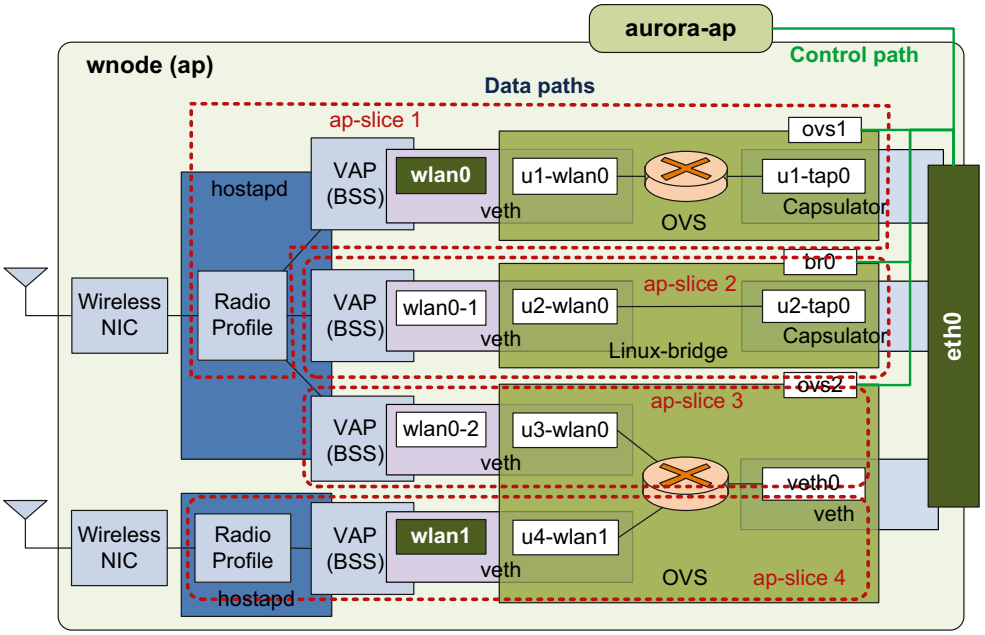

Fig. 7 Aurora-AP Datapath Virtualization

virtual interfaces. Aurora-AP includes plug-ins for different programs that behave like bridges inside the Linux operating system such as Open vSwitch [25] and the basic Linux bridge. The *Open vSwitch* (OVS) software allows tenants to create a virtual switch with multiple virtual interfaces attached as ports. The OVS also supports the OpenFlow protocol, enabling tenants to control the traffic flows of their slices using an OpenFlow controller. The OVS plug-in is a key technology in Aurora-AP that enables wireless flow and network virtualization.

(3). *Virtual WiFi radio interfaces*: In *alix3d2* APs, up to two radio cards are supported and allocated as separate resources running separate *hostapd* instances. Aurora-AP deploys an 802.11-specific module that allows tenants to configure the wireless radio parameters of the radio interfaces. It can also create virtual radio interfaces with which the virtual interfaces and virtual bridges can interact with. The main flavor in Aurora-AP is OpenWrt, which is an open-source Linux-based firmware for wireless access points and wireless routers [20]. The concept of *base BSS* and *guest BSS* is applied over OpenWrt by Aurora-AP. A single base BSS is instantiated per radio card and physical radio profile. Additional allocations are guest BSSes that must share the same radio profile as the base BSS. This limitation only applies to radio nodes without protocol-based virtualization.

The detailed parsing, validation and generation of slice configuration blueprints are delegated to individual modules and plug-ins. As such, modules and plug-ins can be added or removed without affecting the rest of the Aurora framework. This also means that the Aurora software can easily operate on different types of resource nodes by using different plug-ins to match with the specific technologies available on the node. In this way, Aurora satisfies the requirement as a common platform for different virtualization perspectives and technologies.

5.2 Deployment of aurora in OpenStack and SAVI

The Aurora service components are relatively independent from SAVI and other OpenStack services. Of course, there are different points of integration between the Aurora service components and SAVI/OpenStack in order to provide a unified infrastructure-wide end user experience. A basic overview of the deployment of Aurora inside SAVI [12] and OpenStack [2] is shown in Fig. 8, assuming readers are familiar with [2] and [12].

(1). *Tenant and user authentication*: All commands issued by users through the Aurora-Client must be validated using a token-based authentication system managed by the OpenStack Keystone service [26]. The authentication component is used to identify the user and verify what capabilities that user has with the Aurora service components within a tenant project. The validation token is then passed to Aurora-Manager, which is co-located with all other service managers (Nova, Swift, etc.) on the SAVI edge controller.

(2). *Resource database*: The information about the state of wireless resources and their relationship is stored in the MySQL database on the edge controller, along with other OpenStack and SAVI services. In such a way, the

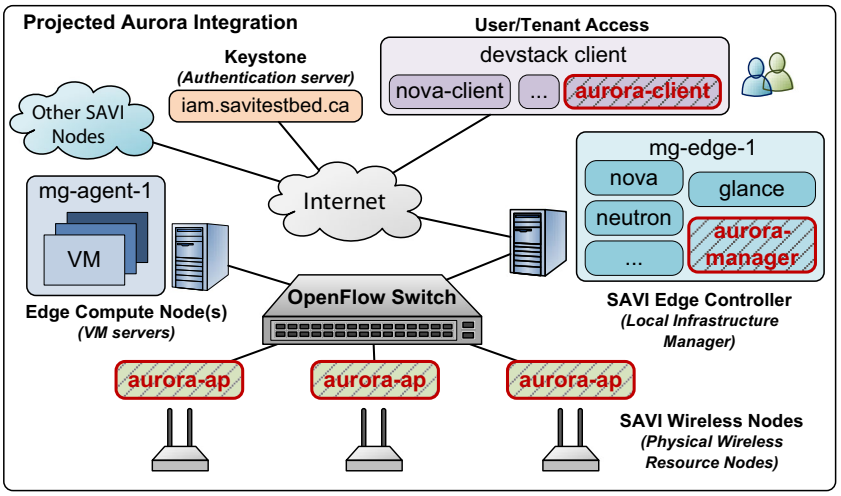

Fig. 8 Deployment of Aurora Inside SAVI/OpenStack

Aurora resource database can allow other services to access information about wireless resources.

(3). *Network connectivity*: The most important aspect of the integration between Aurora and SAVI is the network connectivity between the wireless resource nodes and the rest of the testbed infrastructure. The physical APs are connected to the SAVI edge node switch through Ethernet. Aurora relies on OpenStack Neutron to connect the wireless slices on physical nodes to virtual networks owned by their corresponding tenant.

5.3 How to extend the aurora framework

The creation of new plug-ins and modules inside Aurora is relatively simple due to the way Aurora is modularized and does not significantly increase development overhead compared to writing a customized automation script. The basic steps are outlined as follows:

(1). *Implementation of component technologies* (or *resource components* in Aurora terminology): First, the implementation of specific components needed for virtualization is required. This step is the same whether or not the Aurora framework is present and can be developed independently from Aurora. Even though there is no enforcement by Aurora, it is still suggested to modularize these implementations to match with the modularity of the Aurora framework in order to maximize the future reuse of the components by other tenants.

(2). *Creating plug-ins for Aurora*: This is a new step introduced if these components are to be integrated into Aurora. The developer tenant must write wrappers to interface the Aurora-Agent and Aurora-Manager with their new components. The equivalent steps taken when Aurora is not used would be the creation of automation scripts to deploy their architecture. The development of wrappers, which include full component APIs, is generally a more modularized and organized method compared to the composition of scripts.

(3). *Testing plug-ins for Aurora*: In order to validate the implementations as well as the new plug-ins and modules, native testing (i.e., with physical access point hardware) and debugging of the Aurora framework itself are required. Such a testbed can be assembled using resources on the SAVI testbed within a slice of the infrastructure, in the same way Aurora-AP is currently deployed.

(4). *Deployment of architecture*: At last, the custom wireless virtualization architecture can be deployed over the virtualized infrastructure. In the case without Aurora, it is hardly possible to run multiple wireless virtualization architectures at the same time over the same infrastructure. With Aurora, the resources and components connection topology is fully defined in a centralized blueprint file, facilitating the deployment and customization of an architecture. At the same time, some components such as Capsulator already exist as Aurora plug-ins. These components can be directly used by the blueprint file without much setup overhead by the tenant.

6 Conclusion

Aurora is mainly based on open-source implementations of various technologies and is itself open to extensions developed by a tenant. The advantage of running virtualization projects under Aurora is the fact that other pre-written components are available for maximum technology reuse. Writing new modules and plug-ins is encouraged as it is a more

organized way of implementing new virtualization technologies. At the same time, these new components are automatically standardized under a common framework, in turn enabling other tenants to reuse them to build their own slice and maintaining a good ecosystem for research and development. Thus, this software-defined approach to wireless virtualization is a viable way to keep up with the rapidly evolving wireless ecosystem.

Overall, this article only presents half of the picture, mostly focusing on the virtualization of the individual radio resource nodes and the skeleton of the framework. The management of virtualized wireless networks on an infrastructure-wide level, such as the mobility management and dynamic provisioning of resources, is the other half of the picture and can be the subject of further research. Furthermore, advanced scheduling and traffic shaping techniques can be implemented within Aurora as extensions to the slice creation core engine. The integration of SDR platforms should also be considered in order to implement full protocol virtualization and spectrum virtualization. Advanced radio technologies such as cognitive radios and distributed antennas are also potential targets for integration with Aurora. Of course, they will require the design of a different "flavor" of Aurora-Agent. Finally, cellular technologies were not extensively mentioned in this article. However, the architecture of the cellular network can be mapped to the Aurora framework. For instance, the packet core can be implemented on tenant VMs (Aurora-Tenant), whereas the base-stations can be interfaced with a cellular technology-specific Aurora-Agent.

Acknowledgments This work was partially supported by the Natural Sciences and Engineering Research Council (NSERC) through the NSERC Strategic Network for Smart Applications on Virtual Infrastructure (SAVI), and the *Fonds québécois de la recherche sur la nature et les technologies* (FQRNT) via a scholarship.

References

1. McKeown N et al (2008) OpenFlow: enabling innovation in campus networks. SIGCOMM Comput Commun Rev 38:69–74. doi:10.1145/1355734.1355746
2. Baset S (2012) Open source cloud technologies. ACM Symposium on Cloud Computing, In, p 28
3. Bansal M et al. (2012) OpenRadio: A Programmable Wireless Dataplane. In: Proceedings of HotSDN, pp 109–114
4. Attar A, Li H, Leung V (2011) Green last mile: how fiber-connected massively distributed antenna systems can save energy. Wirel Commun 18:66–74
5. Naudts B et al. (2012) Techno-economic analysis of software defined networking as architecture for the virtualization of a mobile network. In: European Workshop on Software Defined Networking, pp 67–72
6. Li L-E, Mao Z-M. and Rexford J (2012) Towards Software-Defined Cellular Networks. European Workshop on Software Defined Networking (EWSDN), pp 7–12
7. Galis A et al. (2009) Management and service-aware networking architectures (MANA) for future Internet—Position paper: System functions, capabilities and requirements. In: Fourth International Conference on Communications and Networking in China, pp 1–13
8. Paul S, Jianli P, Raj J (2011) Architectures for the future networks and the next generation internet: a survey. Comput Commun 34:2–42
9. Sachs J, Baucke S (2008) Virtual radio: a framework for configurable radio networks. In: Fourth Conference on Wireless Internet, pp 61
10. Yang M et al (2013) OpenRAN: a software-defined ran architecture via virtualization. Proc ACM SIGCMM 2013:549–550
11. Wen H, Tiwary P, Le-Ngoc T (2013) Current trends and perspectives in wireless virtualization. In: Mobile and Wireless Networking, pp 62–67
12. Kang J, Bannazadeh H, Leon-Garcia A (2013) SAVI testbed: Control and management of converged virtual ICT resources. In: Integrated Network Management, pp 664–667
13. Wen H, Tiwary P, Le-Ngoc T (2013) Wireless Virtualization. Springer International Publishing
14. Hoffmann M, Staufer M (2011) Network virtualization for future mobile networks: General architecture and applications. In: International Conference on Communications Workshops, pp 1–5
15. Serrano P et al. (2009) A MAC layer abstraction for heterogeneous carrier grade mesh networks. In: Proceedings of the ICT-Mobile Summit
16. Zhou Y et al (2013) Incremental Re-embedding scheme for evolving virtual network requests. IEEE Commun Lett 17:1016–1019
17. Python Programming Language. (2014) http://www.python.org/. Accessed 11 February
18. MySQL, (2014) The world's most popular open source database. http://www.mysql.com/. Accessed 11 February
19. PC Engines alix3d2 product file. (2014) http://www.pcengines.ch/alix3d2.htm. Accessed 11 February
20. OpenWrt Wireless Freedom. (2014) https://openwrt.org/. Accessed 11 February
21. Aboba B (2003) Virtual Access Points. In: IEEE 802.11-03/154rl. https://mentor.ieee.org/802.11/dcn/03/11-03-0154-00-000i-virtual-access-points.doc.
22. Capsulator - OpenFlow Wiki. (2014) http://archive.openflow.org/wk/index.php/Capsulator. Accessed 11 February
23. Yap K et al. (2010) Blueprint for introducing innovation into wireless mobile networks. In: Second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures, pp 25–32
24. Dely P et al. (2012) Cloudmac—an Openflow based architecture for 802.11 MAC layer processing in the cloud. In: Globecom Workshops, pp 186–191
25. Open vSwitch An Open Virtual Switch. http://openvswitch.org/. Accessed 11 February 2014
26. Keystone Middleware Architecture. (2014) http://docs.openstack.org/developer/keystone/middlewarearchitecture.html. Accessed 11 February