dVirt: a Virtualized Infrastructure for Experimenting BGP Routing

Iuniana Oprescu*† , Mickaël Meulle*
*Orange Labs, France
{mihaela.oprescu, michael.meulle}@orange-ftgroup.com

Philippe Owezarski†‡
†CNRS ; LAAS ; 7 avenue du colonel Roche,
F-31077 Toulouse Cedex 4, France
‡ Université de Toulouse ;
UPS, INSA, INP, ISAE ;
UT1, UTM, LAAS ;
F-31077 Toulouse Cedex 4, France
owe@laas.fr

***Abstract*—In this paper we propose dVirt, a distributed virtualized infrastructure enabling network operators to better grasp the routing mechanisms and anticipate network behavior. dVirt offers an emulated layer 2 connectivity that allows accurate simulation of IP routing and protocol dynamics. Using virtualization and Quagga routing software, we achieve scalable simulations while keeping a realistic model of the routing layer.**

To illustrate the benefits of simulating networks with dVirt, we study a well-known simple Border Gateway Protocol (BGP) topology subject to routing oscillations. dVirt allows for quick deployment of this topology and easy monitoring of BGP message exchanges.

***Keywords*-simulation, virtualization, routing, BGP**

I. Introduction

The Internet is a collection of more than 35000 networks called Autonomous Systems (ASes), each AS being under the authority of a specific administrative entity. The Border Gateway Protocol (BGP) is the de facto standard used for communications between the ASes, whereas the routing within a given AS is handled by an Interior Gateway Protocol (IGP).

Initially designed for exchanging reachability information among several research networks, BGP plays a crucial role in the current Internet: it handles inter-domain routing for protocols that support a wide range of aplications with different traffic requirements. Facing an increasing number of Internet users, however, BGP needs to adapt and keep up with demands regarding complex architectures and new features [1].

Transitions to new routing conditions can be complex and unexpected issues may arise. Despite the plethora of software tools available to model and experiment BGP configurations, there is no dedicated tool offering an automated testbed that allows for accurate simulations and interactions with the underlying protocol layers. dVirt federates multiple functionalities into a flexible tool enabling the user to automatically deploy and evaluate a network.

The ultimate goal for dVirt is to be able to "clone" a full ISP network on top of a virtualized infrastructure running on a smaller number of servers. dVirt aims to reproduce the actual events in a network, experiment with the real configurations and addressing schemes. The proposed framework can be used to check correctness (e.g., avoid oscillations), compare convergence time for different setups or even implement and test additional features on top of the existing protocol stack.

dVirt relies on virtualization techniques and routing tools: we use the Xen [2] hypervisor and virtual machines to represent a large number of network equipments and we put to work the Quagga [3] software routing suite for simulating the multiple routing instances. We take advantage of sbgp [4], a simple BGP4 speaker and listener, to mimic the behavior of neighboring ASes by injecting external route advertisements into the edge routers of the simulated network.

II. Related work

Previous inquiry methods fall back into three main categories: real testbeds, simulation and emulation. A network testbed consists of the actual equipment being deployed, configured and dedicated to conducting experiments. Despite the accurate results provided, a real testbed is ill-suited for large scale tests and has other inconvenients such as the cost of the equipment, the time necessary for setting up the testbed and tearing it down after the experiment.

Initiatives such as Cloonix [5], Virconel [6], NetKit [7], AutoNetKit [8], Virtual Network User Mode Linux (VNUML) [9], Virtual Square with its Virtual Distributed Ethernet [10] framework or the Xen-based virtual integrated network emulator viNEX [11] provide tools and general frameworks for simulation or emulation purposes. The objectives of these networking environments are diverse and often target specific aspects of networking, whereas dVirt realistically emulates the entire protocol stack starting with layer 2.

In the category of training platforms, Dynamips [12] and its front-end Dynagen [13] offer an emulation of specific Cisco equipment, running IOS [14] software on commodity hardware. Several instances of virtual routers make it possible to build a small network and run educational experiments.

A different environment consists of emulated networks located on geographically distributed nodes. The shared testbeds offer physical network equipment on which researchers can conduct experiments in a granted slot. A drawback of using

such a testbed is the lack of control in the environment which leads to unpredictable and unrepeatable experiments.

Among the many BGP tools available (see for example [15]), the C-BGP solver [16] shows some limitations due to the complexity of the model. C-BGP takes into account the logic of the BGP decision process and can compute the outcome of the path selection mechanism for networks of several routers. It is possible to build a model of a large network and investigate changes in the routing or in the topology of such networks. The tool's purpose does not include handling the TCP connections, packet exchanges between the peers or BGP timers such as the MRAI.

Developed by J. Qiu as a lighweight event-driven simulator, simBGP [17] is able to perform large scale simulations while ignoring the underlying protocols such as TCP or IP. Although protocol dynamics can be investigated, simBGP does not offer the full environment for an accurate study of the whole communication mechanism.

III. dVirt: a BGP Simulation Framework

dVirt is a software library written in Python for automatically deploying a given BGP network. It can be controlled from a single machine through simple and flexible inputs that avoid the individual provisioning of resources and configuration of routing protocols. The typical use of dVirt is to simulate the entire topology of one or more large ISP networks and incorporate realistic configurations.

A. dVirt Overview

Compared to previous tools, dVirt is a heavier simulator but removes many barriers thanks to its full customization. dVirt relies on open-source software and runs real operating systems, it supports many real network conditions such as addressing, multitasking of the routing processes and inter-protocol interactions.

dVirt creates an Ethernet topology of virtual machines (VMs) running on top of hypervisors that are mutually reachable at the IP layer. dVirt emulates virtual point-to-point Ethernet connectivity between pairs of router interfaces using virtual switches provided by the Open vSwitch software. Open vSwitch enables virtual Ethernet connectivity between two routers located either on a single or two distinct hypervisors by encapsulating Ethernet traffic inside GRE tunnels.

The OSPF and BGP topologies are automatically configured to enable full reachability inside each AS and setup (mono-hop) eBGP sessions. Routers are running the Quagga routing software with OSPF and BGP daemons to simulate the demanded network. External neighbors of the deployed topology are emulated using sbgp software instances running in one or more additional VMs.

dVirt emulates the full protocol in each router and allows the study of the BGP protocol dynamics by directly running Quagga with all the implemented features. The tool can also be used to deploy modified versions of the Quagga software and therefore handles many routing protocol testing scenarios.

dVirt simplifies the instrumentation of experiments conducted according to a simulation scenario. The user can directly use python bindings to execute existing or user-defined functions. Network monitoring functions run on routers and provide information about the state of the BGP routers, allowing thus to obtain exact measurement data.

B. dVirt Management Network

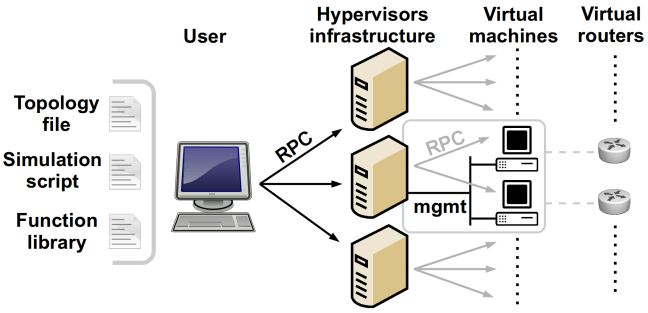

Fig. 1. An overview of the dVirt components: the user interacts through the RPC with the remote hypervisors and the corresponding virtual machines.

To allow permanent communication between the user and the routers, dVirt separates the infrastructure in two distinct networks: a management network for remote access and a test network for the actual simulation. Each VM has a local IP address configured on the interface attached to the management bridge defined on each hypervisor as seen in Fig. 1.

To interact with the remote routers, dVirt provides two libraries to exchange files and do remote calls: SSH and RPC. The SSH library allows to exchange files and send commands to the VMs over an ssh connection with text output.

The RPC (remote procedure call) library enables the creation of a TCP tunnel between the user and any hypervisor in order to execute requests directly on the hypervisor with a simple function call. The output is a python object that is serialized and sent back to the user over the TCP session. The dVirt RPC library has the particular ability to allow transparent execution of RPC requests from the user to a VM or router. The RPC resorts to the hypervisor as an intermediate point that forwards the request in an embedded call directed to the virtual router, as shown in Fig. 2.

dVirt comes with a set of pre-defined functions for the RPC server-side for interaction with hypervisors, VMs, routers and their installed software. The user can easily improve the existing library by adding new functions to the python files in the library. During the next deployment, dVirt will automatically update the RPC library of each hypervisor and each VM making available the new user-defined functions.

C. Virtual Routers

dVirt relies on the Xen open-source software to achieve the **virtualization** of x86 CPU architectures. The Xen hypervisor allows one physical machine to run multiple router instances

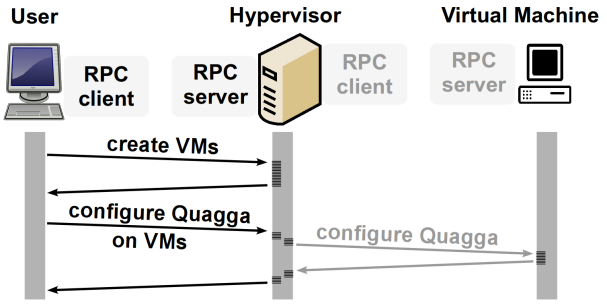

Fig. 2. The user launches a request that can be executed either by the hypervisor or the virtual machines. Note that for the calls on VMs, an embedded request is forwarded by the hypervisor to the corresponding VM.

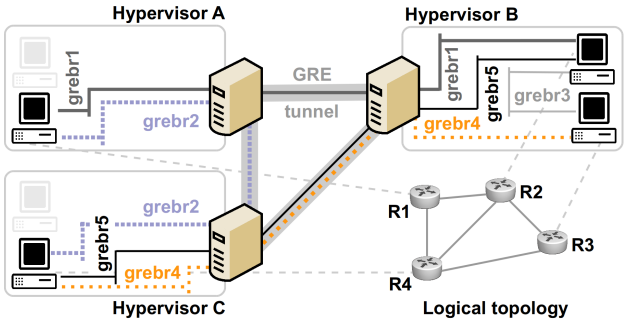

Fig. 3. An overview of the communication in point-to-point mode: the bridges that have the source-destination pair on distinct hypervisors will be encapsulated in the GRE tunnel between the hypervisors.

by acting as an abstraction layer to the bare hardware and isolating the virtual machines from the external networks.

Xen handles the concurrent access of the VMs to the resources and manages the execution of the guest OSes. In Xen terminology, $Dom0$ is the first operating system that boots automatically and receives privileged rights regarding hardware access and management. From the $Dom0$, the administrator can launch new virtual machines, called $DomU$s and manage all the existing guest machines. dVirt uses $virsh$ management interfaces of the libvirt [18] API to create machines from a customizable xml file where memory and CPU allocation can be changed for any router.

By default, each VM in dVirt is a router or it hosts sbgp software instances to simulate external BGP neighbors. Each VM has a dedicated SWAP filesystem, a CPU, a dedicated memory of 512 MB, and a generic pre-installation of the Linux Debian Lenny operating system (distribution 2.6.26-2-xen-686) customized with the required software such as Quagga or Python.

D. Virtual Ethernet

In the $Dom0$ of each hypervisor, dVirt configures the management network through $virsh$ and uses Open vSwitch to emulate point-to-point links in the experimental network. Quagga runs as a regular application on each virtual machine and takes over the kernel routing of the virtual machine. As seen in Fig. 3, for a pair of source-destination routers, two scenarios are possible: if the routers run on the same hypervisor, they interconnect through a dedicated virtual switch (e.g., R2 and R3 linked with grebr3 on hypervisor B); otherwise the two ends of the link are on distinct hypervisors and dVirt needs to define two Open vSwitches, one for the test interface of each router. The traffic between the routers is then transparently forwarded inside a GRE tunnel (e.g., R1 and R2 connect respectively to grebr1 on hypervisor A and B).

A GRE tunnel is setup between two hypervisors only if two distant routers share a point-to-point link. Multiple links between the same two hypervisors can take the same tunnel since isolation is guaranteed by Open vSwitch.

E. Simulated BGP Network

OSPF handles routing within the AS, whereas BGP interconnects different ASes through external BGP (eBGP) sessions. OSPF achieves full intra-AS reachability and external neighbors are directly connected on the specified interfaces, later redistributed inside the OSPF network.

For inter-domain routing, Quagga implements many BGP features, going from different types of BGP sessions (iBGP or eBGP, route reflector or route-server) to ACLs, filtering, prefix aggregation, etc. During the test phase, the routers are fully capable of forwarding traffic, performing the BGP best path decision process as well as receiving or sending OSPF and BGP protocol messages.

F. dVirt Typical Usage

dVirt requires privileged access to a set of hypervisors running Xen with pre-installed software (an SSH server, Python interpreter, Open vSwitch, Quagga and libvirt).

To automatically deploy the testbed, dVirt needs as an input a topology file describing the actual network. Table I illustrates the elements for defining the simulation: routers, links, BGP sessions. dVirt uses the specified attributes to instantiate a VM for the router RR3, with all the required configuration parameters (distinct management and test addresses, BGP loopback and AS number, etc.). A link between the routers RR3 and Rc is another entry in the topology file, just as the different types of BGP sessions with the desired options.

Once a network has been deployed with dVirt, the user can perform specific tasks by running customized code: simulate network events such as incoming routes, link failures, etc. It is possible to run any software application or traffic generator on any of the existing virtual routers or in additional virtual machines. Opposed to most of the existing BGP simulators or emulators, dVirt does not restrict the set of potential experiments on top of the deployed BGP network.

By default, dVirt simulates external BGP neighbors of routers with the sbgp software. One or more dedicated virtual machines can host sbgp software instances, where each instance emulates one external BGP neighbor. Sbgp can inject BGP routes from a customizable mrt file but dVirt also includes functions to randomly generate routes.

Another feature of dVirt is that it enables different monitoring strategies (tap the traffic over network interfaces, query the Quagga routing daemon periodically or call functions through the command line) to collect protocol and router behavior data.

IV. EXAMPLE

We setup an example BGP topology to illustrate the dVirt framework. Fig. 4 depicts a classical example of a "no solution" topology that permanently oscillates [19].

Consider a route to a given prefix ρ advertised by Rx, Ry and Rz. The border routers Ra, Rb and Rc will prefer their direct eBGP path. Due to the specific topology and the IGP metrics on the links, the route reflectors RR1, RR2 and RR3 will never reach an agreement about the best path to ρ.

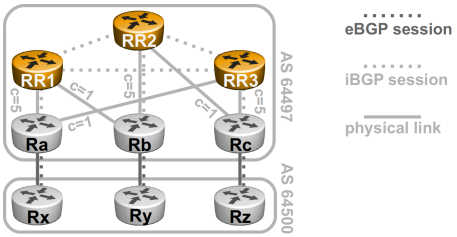

Fig. 4. Topology of experimentation.

To setup the routers in the topology, the user provides the input file containing the description of the BGP network: routers, links and sessions.

Once all the VMs are instantiated and the Quagga routing software is configured, the simulation can start with the injection of prefixes into the border routers Ra, Rb and Rc using three sbgp instances simulating Rx, Ry and Rz.

The routing oscillation from Fig. 4 can be observed with dVirt. Indeed, each of the RRs has one client, but the IGP configuration makes RR1 prefer RR2's client, Rb; RR2 prefers RR3's client, Rc; RR3 prefers the client of RR1, Ra due to lower metrics. Initially, all RRs know the route advertised by their own client and they advertise it to their peers. But as soon as they each receive the routes from their peers, they select as best path the one advertised by the peer and hence they each withdraw their own path. Simultaneously, the neighbor withdraws in its turn the path it had advertised, so the current best path becomes unavailable. Every RR switches back to its own client route and the situation continues indefinitely.

We analyze the messages exchanged between the routers and find that RR3 keeps updating and withdrawing its advertised routes as seen in Fig. 5.

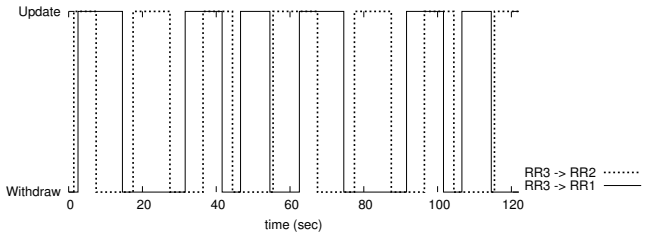

Fig. 5. Route oscillation as seen on RR3: the messages sent by RR3 to its peers, RR1 and RR2.

TABLE I
CONFIGURATION ELEMENTS IN THE TOPOLOGY FILE

[RR3]	[RR3-RC_link]
type=vm	type=vm_link
hypervisor=A	src=rr3
bridge_ipaddress=10.0.0.104	dst=rc
router_id=203.0.113.198	src_ip=203.0.113.98
name=rr3	dst_ip=203.0.113.97
as=64497	netmask=255.255.255.252
bgp_scantime=5	cost=5
	cost2=5
[RR1-RR3]	[RR3-RC]
type=bgp_session	type=bgp_session
src=rr1	src=rr3
dst=rr3	dst=rc
session_type=ibgp	session_type=rr
mrai=10	mrai=0

V. CONCLUSION

In this work we present dVirt, a distributed virtualized testbed for experimenting network scenarios related to BGP routing. We implement features allowing for automated deployment of a BGP topology over a virtual Ethernet. Each BGP router is emulated by routing software in a dedicated virtual machine and links are emulated through virtual switches.

To the best of our knowledge, dVirt is the first BGP simulator able to keep an accurate model of the protocol stack. It can also reproduce routing dynamics, replay network traces in realistic conditions and take into account protocol interactions between OSPF and BGP as illustrated in the example of an oscillating topology.

REFERENCES

[1] V. V. den Schrieck, P. François, and O. Bonaventure, "BGP Add-Paths: The scaling/performance tradeoffs," *IEEE Journal on Selected Areas in Communications*, vol. 28, no. 8, pp. 1299 – 1307, 2010.

[2] The Xen Project, http://www.xen.org/.

[3] K. Ishiguro, "Quagga Software Routing Suite," http://www.quagga.net/.

[4] C. Labovitz and M. Hirabaru, "MRT: Merit's Multi-Threaded Routing Toolkit ," http://mrt.sourceforge.net/.

[5] Cloonix, http://cloonix.net/.

[6] Y. Benchaïb and A. Hecker, "VIRCONEL: A new emulation environment for experiments with networked IT systems," in *Proc. of High Performance Computing and Simulation Conference (HPCS)*, 2008.

[7] Netkit, http://wiki.netkit.org/index.php/Main_Page.

[8] S. Knight, H. Nguyen, M. Roughan, N. Falkner, O. Maennel, and R. Bush, "How to build complex, large-scale emulated networks," *Proc. of TridentCom*, 2010.

[9] F. Galán, D. Fernández, W. Fuertes, M. Gómez, and J. López de Vergara, "Scenario-based virtual network infrastructure management in research and educational testbeds with VNUML," *Annals of Telecommunications*, vol. 64, pp. 305–323, 2009.

[10] L. Bigliardi, "Design e implementazione del nuovo framework Virtual Distributed Ethernet: analisi delle prestazioni e validazione sulla precedente architettura," Master's thesis, Università di Bologna.

[11] A. M. Mukwevho, J. A. van der Poll, and R. M. Jolliffe, "A Virtual Integrated Network Emulator on Xen (viNEX)," in *Proc. of SIMUTools International Conference on Simulation Tools and Techniques*, 2009.
[12] C. Fillot, "Dynamips," http://www.ipflow.utc.fr/index.php/Cisco_7200_Simulator.
[13] G. Anuzelli, "Dynagen," http://www.dynagen.org/.
[14] Cisco IOS, http://www.cisco.com/.
[15] The Border Gateway Protocol Advanced Internet Routing Resources, http://www.bgp4.as/tools.
[16] B. Quoitin and S. Uhlig, "Modeling the routing of an autonomous system with c-bgp," *IEEE Network*, vol. 19, no. 6, November 2005.
[17] J. Qiu, "SimBGP : Python Event-driven BGP simulator," http://www.bgpvista.com/simbgp.php.
[18] Libvirt - The virtualization API, http://www.libvirt.org/.
[19] T. G. Griffin and G. Wilfong, "On the correctness of iBGP configuration," *SIGCOMM Computer Communications Review*, vol. 32, no. 4, pp. 17–29, 2002.