Laboratorio Remoto Virtual para la Experimentación con TCP/IP

Working Paper · April 2017

1 author:

Aldo Abel Crespo
Universidad Nacional de La Pampa
3 PUBLICATIONS **0** CITATIONS

Laboratorio Remoto Virtual para la Experimentación con TCP/IP

Santiago H. Nicolau, Ricardo A. Furch, Juan C. Hernández, Lucas Wals Ochoa y Abel Crespo
Facultad de Ingeniería Universidad Nacional de La Pampa, shnicolau@yahoo.com.ar

***Resumen*— Un diseño innovador de una aplicación de laboratorio virtual para la experimentación con protocolos y servicios bajo TCP/IP, debe contar con las siguientes características: *i)* ser multiusuario, *ii)* la aplicación debe basar su funcionamiento en un modelo de servicio del tipo cliente/servidor, *iii)* el laboratorio debe ser accesible en forma remota desde clientes web ejecutados bajo cualquier tipo de SO, *iv)* el conjunto de software que conforme la aplicación se debe basar en software de fuentes abiertas, *v)* la solución debe emplear una técnica de virtualización ligera para un despliegue eficaz e inmediato de máquinas y dispositivos virtuales, *vi)* los usuarios deben poder construir en forma gráfica escenarios de experimentación que involucren la interconexión de computadoras, conmutadores Ethernet, y *routers, vii)* los usuarios deben poder almacenar localmente, datos y/o archivos de interés generados en el ámbito de la aplicación "lado servidor" del laboratorio virtual, y; *viii)* la aplicación no debe presentar problemas de incompatibilidad por incumplimiento de dependencias de software (desde la perspectiva del cliente que accede al laboratorio virtual). Este artículo presenta una herramienta que cumple con todos los requisitos mencionados en el resumen.**

***Palabras clave*—Laboratorio, Remoto, Contenedores, Linux, SDN.**

I. INTRODUCCIÓN

Desde hace más de una década, las cátedras de Redes y Comunicaciones en la Facultad de Ingeniería de la UNLPam utilizan aplicaciones bajo GNU/Linux para virtualizar máquinas, conmutadores Ethernet, y *routers*. Ello ha permitido la construcción de complejas topologías de red para experimentar con una gran variedad de protocolos y servicios bajo TCP/IP. El lector, debe notar que para lograr el mismo objetivo en una infraestructura de laboratorio basada en componentes reales, se requiere de un gran número de recursos de hardware, que en el caso general, exceden las posibilidades presupuestarias de la gran mayoría de las instituciones educativas universitarias. Por la última razón descrita, existe una variada colección de herramientas de software destinadas a virtualizar máquinas (computadoras virtuales) y componentes de conmutación en capas 2 y 3 del modelo de referencia OSI (L2 y L3) para interconectarlas en red [1], [2].

A continuación se realizará una breve síntesis de las opciones más relevantes, utilizadas intensivamente en asignaturas relativas al área de las redes de computadoras en universidades de todo el mundo.

Netkit [3] fue elaborado por el área de Redes de Computadoras de la Universidad de Roma. Básicamente se trata de una serie de scripts escritos en *bash*, para ejecutar, detener, eliminar, y listar computadoras virtuales. Utiliza *"User Mode Linux"* (UML) [4], [5] como tecnología de virtualización. UML, necesita de una versión del núcleo Linux que debe ser convenientemente configurado y compilado para la arquitectura UML. Luego se debe construir un sistema de archivos para instalar programas y aplicaciones, de acuerdo a las temáticas a experimentar y por último se deben instalar las utilidades UML que provee entre otras herramientas, un conmutador de tramas Ethernet. Proyectos de software como NetGui y jNetEdit, utilizan los scripts provistos por Netkit para implementar interfaces gráficas de usuario basadas en Java.

Common Open Research Emulator (CORE) [6], es una aplicación basada en software de fuentes abiertas, desarrollado por la división "tecnologías de red" de la empresa Boeing, en colaboración con el Instituto Naval de Investigación en los EEUU. CORE, fue implementado en el sistema operativo FreeBSD y permite emular complejas topologías de red. Utiliza una técnica de virtualización liviana basada en jaulas *(jails)*, cada una de ellas caracterizadas por cuatro elementos: *i)* un sub árbol de directorio; *ii)* un nombre de host; *iii)* una dirección IP; y, *iv)* un comando, o nombre de un archivo ejecutable en la jaula. CORE ofrece al usuario una interfaz gráfica (GUI) construida con el lenguaje de programación TCL junto a un kit de herramientas para uso gráfico provisto por TK (TCL/TK).

Mininet [7] es producto de un proyecto generado en la Universidad de Stanford, con un gran nivel de actividad en la actualidad. Utiliza contenedores virtuales basados en LinuX Containers (LXC) [8]. La tecnología LXC es un tipo de virtualización a nivel del núcleo Linux, en el que cada archivo de sistema se almacena en entornos aislados entre sí, y cualquier proceso en un contenedor no tiene relación con recursos o procesos fuera del propio contenedor. Bajo esta tecnología, todos los sistemas virtuales comparten el núcleo del sistema operativo en el host anfitrión, de modo que los entornos virtuales generados perciben un desempeño cercano al de las aplicaciones nativas en el propio host. Para proveer conectividad entre máquinas virtuales (contenedores) y crear redes definidas en software distribuidas sobre un conjunto de servidores físicos, Mininet utiliza Open vSwitch (OVS) [9] como tecnología de interconexión. OVS es una implementación en software de un conmutador Ethernet, que soporta una amplia variedad de protocolos en capa de enlace de datos, y el protocolo OpenFlow [10] para un encaminamiento flexible y de alto desempeño en redes definidas por software (*Software-Defined Networking)*. Los laboratorios virtuales en Mininet, pueden ejecutarse según dos modos; *i)* desde línea de comandos, y; *ii)* vía una API desarrollada en el lenguaje de programación Python.

Todas las aplicaciones de software mencionadas en los párrafos previos de esta sección, son del tipo *"stand alone"*. Ello significa que la instalación se debe replicar en cada una de las computadoras que conforman un

laboratorio de redes. En el caso de UML, utiliza una tecnología de virtualización que aumenta la penalidad en lo que a medidas de desempeño se refiere. CORE y Mininet se asemejan respecto a la tecnología de virtualización que utilizan, aunque Mininet provee mayor aislación entre las instancias virtuales generadas (contenedores).

Una opción superadora de laboratorio virtual, es VirLabNet, cuyo diseño e implementación, responde a las características mencionadas en el resumen de este artículo.

A continuación, en la sección II se presentará una síntesis de la arquitectura de laboratorio virtual remoto propuesta. Posteriormente en la sección III se describirá la interacción de usuarios con el laboratorio virtual. La sección IV tratará sobre la interfaz gráfica de usuario, utilizada para: crear, modificar o eliminar topologías. En la sección V se presentará lo relativo a trabajos futuros, y en la sección IV las conclusiones del artículo. Finalmente el lector encontrará una reseña bibliográfica en la que se basó este artículo.

II. ARQUITECTURA DE LABORATORIO REMOTO

La Fig. 1 muestra un esquema simplificado de la interacción entre los componentes de software que componen la infraestructura de laboratorio virtual (VirLabNet) presentada en este artículo.

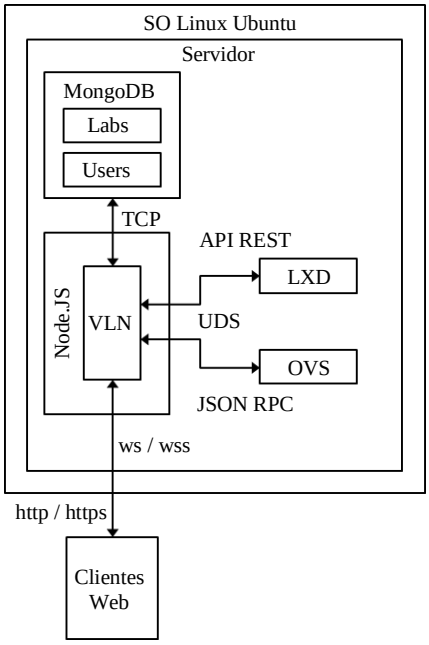

Fig. 1: Síntesis de la arquitectura de laboratorio virtual.

La arquitectura se basa en dos características relevantes: *i)* emplea componentes de software orientado a eventos en el lado servidor de la aplicación, y; *ii)* utiliza un protocolo para la interacción entre cliente y servidor que minimiza el *overhead* y la latencia en el intercambio bidireccional *(full duplex)* de mensajes. En el lado servidor, la aplicación fue desarrollada con el componente de software Node.JS [11] que provee un entorno de ejecución JavaScript, multiplataforma de código abierto, basado en el motor V8 de Google para su cliente web Chrome. Node.JS se caracteriza por ser un sistema de comunicación no bloqueante, y orientado a eventos asíncronos. Estos aspectos hacen de Node.JS una opción ideal para el desarrollo de aplicaciones orientadas a la transferencia de datos de tiempo real. Node.JS efectúa una compilación directa del código fuente JavaScript a código de máquina, evitando operaciones de interpretación del lenguaje y de generación de códigos intermedios *(bytecodes)*, aspecto que se traduce en una optimización en lo que a velocidad de ejecución se refiere. Adicionalmente Node.JS incorpora en forma nativa el manejo y la manipulación de textos con formato *JavaScript Object Notation* (JSON definido en el RFC 7159) empleado para el intercambio de datos, almacenamiento de configuraciones u otros propósitos generales.

Los clientes web´s interactúan con el lado servidor de la aplicación utilizando el protocolo WebSocket (ws/wss) descrito en el RFC 6455. Se trata de una solución eficaz para utilizar una única conexión TCP por cliente para el tráfico bidireccional, evitando el paradigma tradicional de las aplicaciones web basadas en la secuencia estricta *i)* petición de cliente *ii)* respuesta del servidor.

LinuX containers Daemon (LXD) [12] es un hypervisor para contenedores Linux desarrollado por Canonical Ltd. Es un complemento para los contenedores Linux (LXC) que facilita su uso y añade nuevas posibilidades. Brinda un servicio que le permite a un usuario "sin privilegios" la posibilidad de: crear, iniciar, detener, y/o eliminar contenedores basados en una imagen de un sistema de archivos. Muchas de las acciones descritas se pueden realizar desde la línea de comandos, y un valor agregado es el hecho de que consta de una API REST *(Representational State Transfer)* [13] disponible para ser utilizada desde una dirección remota de Internet o bien en el equipo local mediante el uso de *Unix Domains Socket* (UDS) [14]. Una API REST es una interfaz para acceder a recursos identificados mediante *Uniform Resource Identifiers* (URIs), utilizando los comandos o verbos del protocolo http: *i)* GET, *ii)* PUT, *iii)* POST, y; *iv)* DELETE, direccionados a un recurso determinado (URI). Para resolver la interacción entre la aplicación VLN (elaborada en el marco del proyecto objeto de este artículo) y LXD; se desarrolló una librería propia que accede a la API REST publicada por LXD para el intercambio de datos en formato JSON vía UDS (Fig. 1).

OVS es un software de código abierto diseñado para ser utilizado como un conmutador Ethernet. En un ambiente de virtualización, permite el intercambio de tráfico de red entre contenedores virtuales distribuidos entre plataformas de servidores físicos. Ejecuta un demonio *(ovs-vswitchd)* y posee su propia base de datos para almacenar la configuración del conmutador virtual. El protocolo de administración *Open vSwitch Data Base* (OVSDB estandarizado en el RFC 7047) es el encargado de suministrar un acceso programático a la base de datos de OVS.

Para vincular VLN y OVS, se utilizó el protocolo *Remote Procedure Call* (RPC) embebido en el formato JSON (JSON-RPC). RPC utiliza el modelo cliente/servidor, en que el cliente (VLN) envía una solicitud JSON-RPC, y el lado servidor JSON-RPC (en OVS) emite una respuesta. JSON-RPC es una implementación que se caracteriza por ser ágil, simple, sin estado e independiente del protocolo de transporte que se emplee (http, sockets TCP, UDS, etc.); en el ámbito de este artículo se utilizó UDS (Fig. 1).

La capa de persistencia de datos se implementó vía el software MongoDB [15], [16]; un gestor de código abierto multiplataforma de base de datos "no relacional" que se

adapta en forma óptima cuando se trata de almacenamiento de información orientada a grafos.

III. INTERACCIÓN USUARIOS - LABORATORIO VIRTUAL

La interacción de usuario con el laboratorio virtual se realiza a través de la capa de presentación o interfaz gráfica de usuario, diseñada e implementada para ser utilizada desde cualquier tipo de cliente web con soporte del estándar HTML5. La capa de presentación proporciona una región rectangular generada por el elemento *canvas* de HTML5, cuya finalidad es definir un entorno para la creación y/o edición de imágenes, que involucra acciones de los usuarios relativas a; la selección, el arrastre, el desplazamiento, la fijación, y la vinculación entre componentes en el área definido (enlaces). Todas las acciones descritas se implementan utilizando la librería JavaScript Processing.js. Finalmente, el diseño de la interfaz gráfica de usuario involucra los componentes de software de código abierto desarrollado por terceros, tal como la librería JavaScript jQuery que facilita el desarrollo de aplicaciones enriquecidas compatibles con todos los tipos de clientes web. Con el laboratorio en el estado ejecución, cada modificación en la posición de los componentes en el módulo de presentación, se almacena automáticamente en base de datos, de modo que un usuario verá reflejado los cambios efectuados aun cuando detenga y reinicie la instancia de laboratorio virtual.

La Fig. 2 ilustra el esquema de interacción de usuarios con el laboratorio virtual. Tal como se observa en la gráfica, el usuario *"user"* crea una nueva topología o edita una existente, luego: *i)* inicia el laboratorio virtual, *ii)* el *browser* envía la información de la topología al componente VLN (ejecutado por Node.js), *iii)* el componente VLN comprueba la sintaxis de la información recibida, *iv)* sí la sintaxis es correcta, procede a almacenar los datos relativos a la topología en el módulo base de datos, *v)* VLN aguarda la confirmación de la operación de almacenamiento, *vi)* VLN invoca a OVS para crear los conmutadores Ethernet (*vi y vii*); y a LXD para crear e iniciar los contenedores virtuales (*viii* a *xiii*). Finalmente VLN vincula el intérprete de comandos *(shell)* de los contenedores virtuales, con los terminales web´s en la capa de presentación del usuario (*xiv* a *xvi* en la Fig. 2).

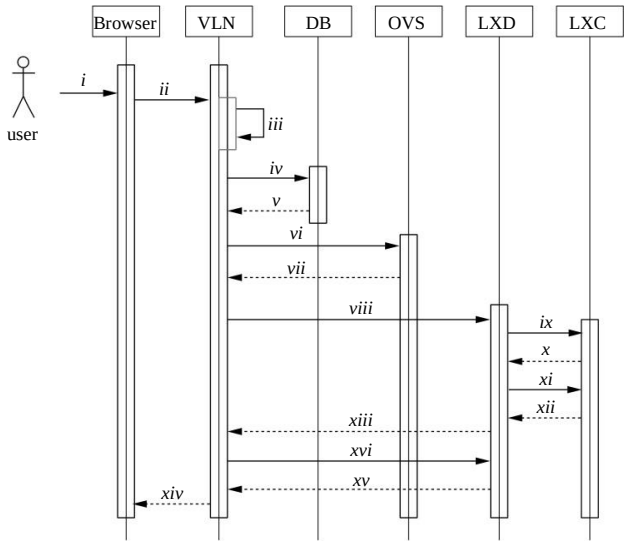

Fig. 2: Diagrama de interacción de usuario con el laboratorio virtual remoto.

IV. INTERFAZ GRÁFICA DE USUARIO

La Fig. 3 ilustra la interfaz gráfica de usuario (redibujada para que el lector tenga una clara apreciación de los iconos y demás elementos que conforman el *"front end"* de la aplicación). Allí se muestra una topología simple que consta de un conmutador Ethernet (SW1), un *router* (R1) y dos computadoras PC1 y PC2.

Presionando los iconos situados en el nivel superior de la capa de presentación, el usuario puede agregar componentes (computadoras, conmutadores Ethernet y *routers*), eliminarlos, moverlos en el área de trabajo y vincularlos *(links)*. Una vez finalizada la construcción, la topología se ejecuta presionando el botón de inicio o se lo detiene presionando el *mismo* botón.

El usuario selecciona una terminal (elemento TERM) en el lado izquierdo del panel para interactuar con un dispositivo o una computadora de la topología creada e iniciada. La Fig. 3 ilustra la consola emergente luego de haber seleccionado el componente R1, y tal como se observa cada dispositivo virtual provee dos consolas por dispositivo etiquetadas como TERM 01 y TERM 02.

El usuario remoto puede capturar pantallas, descargar archivos de interés en su computadora y actualmente se están confeccionando plantillas de prácticas o topologías pre construidas para su resolución. Adicionalmente en el panel superior a la izquierda, presionando "recursos" el alumno puede acceder a material de lectura recomendado por el instructor de la cátedra.

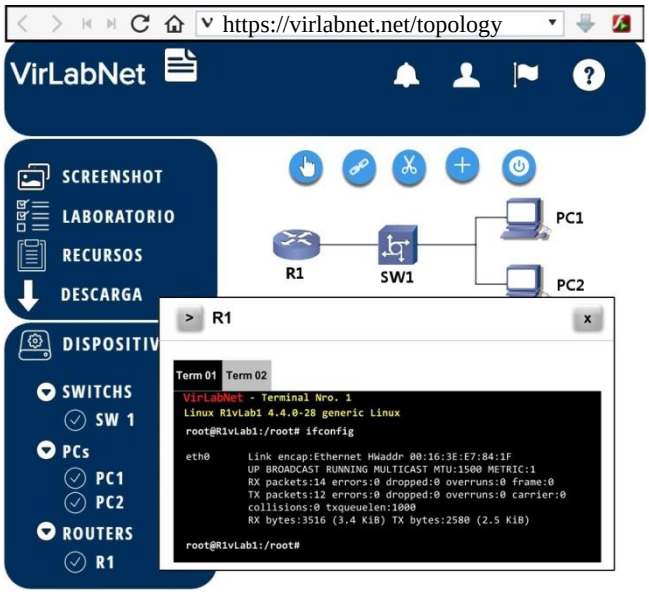

Fig. 3: Interfaz gráfica de usuario (GUI) de la aplicación VirLabNet. Simple construcción de un escenario de red que involucra un *router* (R1), un conmutador Ethernet (SW1) y dos computadoras PC1 y PC2.

V. TRABAJOS FUTUROS

El laboratorio virtual accesible remotamente, para la construcción de topologías que involucran complejos escenarios de red, está funcionando. El modelo operativo responde a la arquitectura mostrada en la Fig. 1, y el diagrama de interacción de usuario con el laboratorio virtual es como se lo representa en la Fig. 2. Los trabajos futuros a implementar tienen que ver principalmente con el componente OVS.

OVS soporta numerosos protocolos y características, sólo comparables a los costosos conmutadores Ethernet utilizados en ámbitos de producción intensivos: *i) Link Aggregation Control Protocol* (LACP), *ii)* IEEE 802.1Q VLAN con *trunking*, *iii) Internet Group Management Protocol* (IGMP) especificado en el RFC 4541, *iv) Shortest Path Bridging* (SPB) de acuerdo al estándar IEEE 802.1aq, *v) Connectivity Fault Management* (CFM) estandarizado por IEEE 802.1ag, *vi) Bidirectional Forwarding Detection* (BFD) descrito por los RFC 5880 y 5881, *vii) Spanning Tree Protocol* (STP, IEEE 802.1D-1998) and *Rapid Spanning Tree Protocol* (RSTP, IEEE 802.1D-2004), *viii)* control fino de *Quality of Service* (QoS) para diferentes aplicaciones de usuarios o flujos de datos, *ix)* políticas de tráfico a nivel de interfaces de máquinas virtuales, *x)* soporte del protocolo OpenFlow, y; *xi)* soporte completo de IPv6 (Internet Protocol versión 6). El lector debe considerar que sólo se han nombrado algunos de los protocolos soportados por OVS.

De acuerdo a lo expresado en el párrafo anterior, es necesario ampliar la API desarrollada, con el propósito de experimentar con la amplia variedad de protocolos soportados por OVS. Ello permitirá realizar experimentos de laboratorio que contemplen los protocolos de capa de enlace de datos en Ethernet descritos en los párrafos anteriores en esta sección.

En la actual versión operativa del laboratorio presentado en este artículo, OVS funciona como un conmutador Ethernet transparente, en el modo almacenamiento y reenvío *(store and forward).*

VI. Conclusiones

El presente trabajo surgió como consecuencia de la necesidad de experimentación en cursos relativos al área de las Redes de Computadoras. En los últimos diez años se utilizó UML como herramienta para la virtualización de computadoras y dispositivos de interconexión de red. La experiencia mostró que cuando se pueden experimentar los conceptos teóricos vertidos en el aula, el proceso de aprendizaje mejora notoriamente y despierta el interés del alumno promedio.

Tal como fue descrito en la sección I de este artículo, UML presenta una elevada penalidad en lo que a desempeño se refiere, penalidad que se acrecienta en escenarios de red que contemplan unas pocas decenas de componentes. Otra desventaja es que se debe instalar el emulador en cada una de las computadoras que conforman un laboratorio de red.

En el contexto señalado y ante la aparición de componentes de software para ser aplicados al área de las Redes Definidas por Software (SDN), este grupo de investigadores encontró la motivación para diseñar un laboratorio virtual, accesible remotamente, utilizando los componentes de fuentes abiertas provistos para la implementación de SDN.

A la solución diseñada e implementada se la denominó VirLabNet, y las principales características respecto de las opciones existentes es que se trata de una herramienta orientada a la web. Los usuarios interactúan con la parte servidora de la aplicación, sin otra necesidad más que la de disponer de una conexión a Internet y un cliente web con soporte de HTML5 ejecutado bajo cualquier sistema operativo.

Una vez construida y ejecutada una topología de red, la respuesta es inmediata desde la perspectiva de usuario (sensación de usuario). Las consolas de las máquinas o dispositivos virtuales se despliegan inmediatamente, y este aspecto es posible por la elección de la técnica de virtualización escogida (LXC), y por los protocolos (en especial ws/wss) y componentes de software orientados a aplicaciones de tiempo real (Node.JS).

La solución presentada en este artículo es inédita. Se realizaron intensas búsquedas en Internet y se comprobó que no existe un proyecto similar en curso.

Tal como se mencionó en la sección V, aún resta fortalecer VirLabNet en lo que se refiere al soporte de protocolos de capa de enlace (L2) en Ethernet. En tal sentido invitamos a la comunidad de Investigadores interesados en sumarse al proyecto para acelerar su desarrollo y posicionarlo en el ámbito educativo, para que lidere el segmento de las herramientas de software destinadas a emular laboratorios para la experimentación con protocolos y servicios bajo TCP/IP.

Referencias

[1] F, Laudares Camargos, Gabriel Girard, and Benoit des Ligneris, “Virtualization of Linux Servers, a Comparative Study”, *en Proc, of the Linux Symposium*, vol. 1, pp. 63-76, 2008.

[2] W. Felter, A. Ferreira, R. Rajamony and J. Rubio, "An updated Performance Comparison of Virtual Machines and Linux Containers," *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 171-172, 2015.

[3] Maurizio Pizzonia, Massimo Rimondini, “Netkit: network emulation for education”, *Software: Practice & Experience, John Wiley & Sons, Ltd*, vol. 46: pp. 133–165. 2014.

[4] Jeff Dike, “User Mode Linux”, *en Proc. of the 5th annual Linux Showcase & Conference Speech Communication*, vol. 5, pp. 3-14, 2001.

[5] Jeff Dike, “User Mode Linux(R)”, *Bruce Perens Open Source Prentice Hall*, 2006.

[6] Ahrenholz, "Comparison of CORE Network Emulation Platforms", *en Proceedings of IEEE MILCOM Conference*, pp.864-869, 2010.

[7] Bob Lantz, Brandon Heller, and Nick McKeown, “A Network in a Laptop: Rapid Prototyping for Software-Defined Networks”. *9th ACM Workshop on Hot Topics in Networks*, 2010.

[8] J. Claassen, R. Koning and P. Grosso, “Linux containers networking: Performance and scalability of kernel modules”, *NOMS 2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 713-717, 2016.

[9] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, “The Design and Implementation of Open vSwitch”, *en Proc. of the 12th USENIX/ACM Symposium on Networked Systems Design and Implementation*, pp. 117-130, 2015.

[10] ”OpenFlow Switch Specification version 1.5.0”, *Open Networking Fundation*, 2014.

[11] Tom Hughes-Croucher and Mike Wilson, “Node: Up and Running”, *O’Reilly Media Inc.*, 2012.

[12] LinuX containers Daemon, site: https://github.com/lxc/lxd.

[13] Roy Thomas Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, *Dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy*, pp. 76-105, 2000.

[14] Stevens, Richard, *“UNIX Network Programming”*, vol. 2, Second Edition. Prentice Hall, 1999.

[15] Kristina Chodorow, “MongoDB: The Definitive Guide”, *O’Reilly Media Inc.*, 2013.

[16] C. Győrödi, R. Győrödi, G. Pecherle and A. Olah, "A comparative study: MongoDB vs. MySQL", *13th International Conference on Engineering of Modern Electric Systems (EMES)*, pp. 1-6, 2015.