Experimenting with Control Operations in Software-Defined Infrastructures

Stuart Clayman, Lefteris Mamatas†, and Alex Galis
Dept. of Electronic and Electrical Engineering, University College London, London, UK
† Dept. of Applied Informatics, University of Macedonia, Thessaloniki, Greece
Emails: s.clayman@ucl.ac.uk, emamatas@uom.edu.gr, a.galis@ucl.ac.uk

***Abstract*—Recent years have seen a trend in network and service infrastructures adopting software networking paradigms. Traditional hardware, such as servers, network equipment and middleboxes, are evolving towards flexible equivalent software technologies: namely virtualized servers, routers and network functions, respectively. At this stage, such Software-Defined Infrastructures (SDI) are mixing traditional with modern solutions. Virtual servers host traditional operating systems, resulting in heavy virtual machines with most of their codebase often unneeded, and also a mismatch between the flexibility capabilities of the new entities and their equivalent management facilities, especially towards providing holistic service-aware capabilities.**

Along these lines, we present an open-source experimentation solution for flexible and service-aware management facilities on top of heterogeneous softwarized network resources, the Very Lightweight Software-Driven Network and Services Platform (VLSP). VLSP exhibits the following properties: (i) it provides a complete integrated management platform for SDI environments as a basis for experimentation; and (ii) it is distributed and scalable through adopting lightweight virtual entities, making it suitable for testing a wide-range of management features over diverse topologies. Here, we give the implementation details of VLSP and experimentally validate its operational advantages.

I. Introduction

Software Defined Infrastructures (SDIs) have advanced in the last few years to address the flexibility requirements of the network infrastructure to meet the proliferation of new network-demanding services and the increasing variety in user and application requirements. For example, a pool of resources can be jointly optimized and utilized on-demand from dynamic services. The difficulties in upgrading hardware equipment and network protocol functionality are being addressed using carefully designed abstractions and open interfaces.

In practice, the softwarization of hardware equipment redrew the line in between hardware and software, allowing better adaptability to the resource constraints and the user requirements. Cloud environments, using virtual machines (VMs) as compute facilities, brought a step change in how management of infrastructure was considered, given its flexibility and dynamic nature. Hardware middleboxes are evolving towards Virtualized Network Functions (VNFs) according to the Network Function Virtualization (NFV) paradigm, allowing them to act as dynamic building-blocks realizing novel services. Furthermore, the rigid architectural constraints of network protocols are being tackled by the Software-Defined Networks (SDNs) performing logically-centralized network control that is decoupled from the data plane, enabling holistic network management.

978-1-4673-9486-4/16/$31.00 © 2016 IEEE

There are many open-source systems in the area of providing and managing SDN and/or NFV environments, such as OpenStack [1], OpenDaylight [2] and OPNFV [3]. OpenDayLight is based on a loosely-coupled architecture, whereby service and virtual network device plugins realize the targeted behaviour based on a common API. OpenOverlayRouter [4] is an edge-oriented solution that instantiates programmable overlay networks and is based on the LISPmob.org code [5]. OpenMANO [6] is an open source implementation of ETSI's reference architecture for Management and Orchestration NFV ISG [7]. On the standardization front, ONF is working on OpenFlow (the prominent SDN protocol) aspects [8] and ETSI [9] is studying the architecture of NFV from the network operators' perspective.

A number of management solutions have been proposed for flexible infrastructures. The CONTENT architecture [10] proposes an orchestrator (at the cloud service layer) federating the IT resources from distributed sites with user-to-data-centre and inter-data-centre multi-layer connectivity services, managed by a SDN-based network layer. The UNIFY consortium [11] devises means to orchestrate, verify and observe end-to-end service delivery from premises and enterprise networks through aggregation and core networks to data centres. The T-NOVA project [12] plans to exploit the concept of NFV allowing operators to deploy VNFs for their own needs, and also to offer them to their customers, as value-added services.

The above proposals provide management facilities focusing on one of the three discussed paradigms mainly (Clouds, NFV or SDN) and other aspects are being provided through relevant extensions. For example, Neutron augments OpenStack clouds [13] with *networking as a service* capabilities, bringing the support of SDN/NFV entities to the Cloud. However, its management facilities are not optimized for a unified and integrated operation of Clouds with SDN/NFV.

Furthermore, some aspects of the new paradigms may be misaligned, mixing new with old technologies. It is common in deployed networks, including cloud and data center networks, to use system-level virtual machines (e.g. Xen, VMware) plus embedded software routers (e.g. Click [14] or Quagga [15]). However, these are heavy for large-scale operation and VM migration capabilities. Our previous experience [16] with such technologies has highlighted some serious issues with such systems. Namely, the number of virtual machines that can run on a physical host is limited, the speed of startup of a virtual

machine can be quite slow, the size of a virtual machine image is quite large, and in terms of highly dynamic networks, and virtualized routers in particular, we observered that 98% of the code functionality was never utilized in any run.

To address the above issue, researchers proposed a number of open-source lightweight virtual servers and routers, such as Mirage OS [17], OSv [18] and ClickOS [19]. These solutions are based on unikernels (or Cloud OS) [20], where the application can be defined with a high-level programming language and its compilation output is a very lightweight and single-purpose virtual machine keeping the essential part of the OS system only (i.e. called for this reason Library OS). These virtual machines may be few megabytes in size and boot up so quickly that they can startup with the establishment of a TCP connection or the reception of a DNS request packet [21]. So far, such proposals are being deployed within existing cloud or SDN environments that are not designed especially for these lightweight virtualized network devices.

From our point of view, the above solutions should be combined together, resulting in high levels of flexibility in Software Defined Infrastructure operations. These unified environments require new efficient and distributed management facilities that are characterized by scalability, reliability, and adaptability to the dynamic conditions, in terms of resource availability and changing service and infrastructure requirements. This shift in approach calls for better, unified, and more efficient management and control with relevant implemented software targeting a number of challenges, such as: (i) efficient service operation, adaptability and deployment, (ii) service and VNF lifecycle automation, and (iii) efficient resource utilization and dynamic scaling (i.e. elasticity).

In order to assist research in this direction, we have developed a fully distributed facility for testing and evaluating the management aspects of such SDIs, the Very Lightweight Software-Driven Network and Services Platform (VLSP), that: (i) integrates our own management and control components along these lines, and (ii) supports lightweight virtual entities. VLSP is an experimentation solution for scalable high-level management features, rather than aiming at production Cloud, SDN or NFV environments. The platform has a simplified view of the whole SDI, as we have abstracted away some of the complex details, but we still retain the same runtime dynamics. On this basis, it allows novel management features to flourish which can then be integrated into production solutions.

VLSP supports the following main capabilities:

Distributed infrastructure implementing logically-centralized management and control: to support optimizations following centralized decisions based on the global view of the system derived using a monitoring infrastructure, i.e. our own integrated monitoring system called Lattice [22].

Experimentation on management facilities: focusing on the experimentation of distributed management and control components of SDI rather than on the data plane performance. Enhanced distributed management, control and monitoring evaluations can be carried out which is difficult to achieve in a running environment using a number of deployed data centers.

Lightweight virtual entity implementation: allowing evaluation of diverse scenarios, including testing of various management lifecycle schemes. The benefits of VLSP over using a hypervisor with virtual machines and standard OS are: (i) better scalability providing lower resource utilization and better resource allocation; and (ii) quicker startup speed and reduced heaviness, eliminating the issue where most of the functionality is not needed.

Last but not least, VLSP supports a number of features assisting in experimentation, such as: (i) visualization tools depicting the topology, the software running on each entity, the network state and the interactions between the VLSP components; (ii) common topological representations and scripts that manipulate topologies based on known distributions, allowing arbitrary topologies to be created; (iii) example virtual entity applications (e.g. for VNFs) with configurable loads in terms of CPU, memory and network utilization; (iv) tailor-made monitoring capabilities supporting information aggregation, custom monitoring probes and adaptability to the monitoring requirements; and (v) a number of virtual entity placement algorithms assisting the optimal data flow establishments.

In this paper, we present the implementation details of VLSP and demonstrate its efficient operation over diverse virtual network topologies. In our paper [23] there is a discussion on the main VLSP architectural artefacts plus a presentation of representative VLSP use-cases. The VLSP open source solution is available at [24] and is not based on any existing software stack, rather the whole system, including a virtual infrastructure manager, a host-level supervisor, and a lightweight virtual entity were designed and built from scratch.

A summary of the remaining paper follows: Section II discusses the VLSP system design and implementation details. Section III shows our evaluation results and discusses a number of different use-cases we investigated experimentally based on the VLSP codebase. Finally, Section IV concludes the paper.

II. The VLSP System

VLSP provides a complete environment and solution for experimenting with scalable high-level management features. It has elements from the service management level down to protocol stack, including a tailor-made monitoring facility. Consequently, we are able to experiment with a new and complete management and control facility over virtual infrastructures based on our *lightweight* virtual entity resembling both servers and routers. The VLSP architecture consists of three main layers to provide a system that is both scalable and modular for Virtual Infrastructure Management. Figure 1 depicts an overview of the architecture. The VLSP layers are:

1) the *Application Layer* which executes Management Applications that define the software components and network functions of a network service together with their configuration parameters and operations.
2) the *Orchestration Layer* which performs most of the management and orchestration activities and is in charge of: (i) managing the full lifecycle operations of the virtual resources in the system, and (ii) allocating the applications running on the virtual nodes.
3) the *Infrastructure Layer* contains both the physical machines hosting the virtual entities and a *Host Controller* which works in a similar way to a standard hypervisor. It

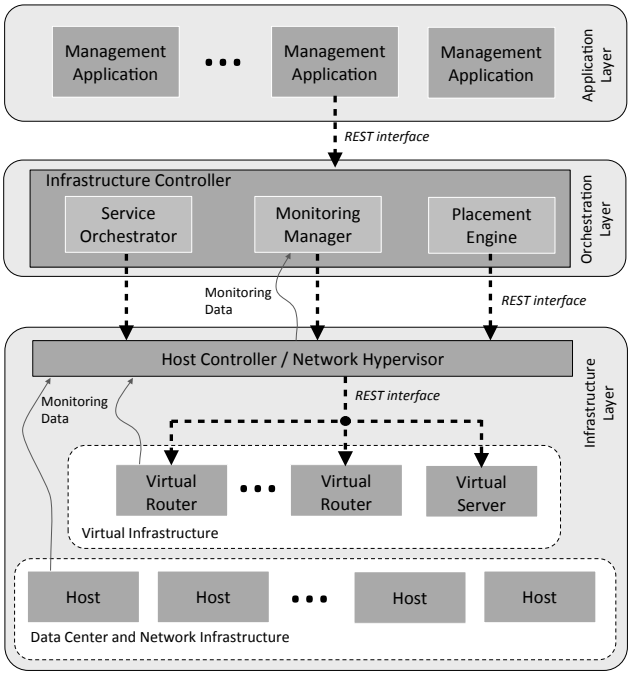

Fig. 1. Overall system architecture and components

manages the virtual resources, and is responsible for the runtime operation of those virtualized resources and the applications they host.

In order to manage the challenging and dynamic nature of virtual infrastructures there needs to be a monitoring system which collects data and reports on the behavior of both the physical resources (e.g. cpu usage, memory usage) and the virtual resources (e.g. virtual link level utilization). The monitoring data is sent to an Infrastructure Controller components, whereby the monitoring information is used to take decisions.

A. Implementation

To validate our design, we created a working implementation of the architecture described above. The VLSP has been implemented by University College London for the purpose of testing and evaluating various aspects of managing Software Defined infrastructures and highly dynamic virtual environments, and is available as an open-source software at [24]. The VLSP consists of the management components plus potentially a large number of virtualized entities, each running inside Java Virtual Machines. The virtual entities execute on a number of physical machines and are independent software elements that communicate with each other via REST interfaces.

To highlight how these components are distributed across the physical resources, consider a setup with 4 hosts. In Figure 2 we depict how these components are placed and how they interact. In the bottom left host the *Infrastructure Controller* is executing. It includes the *Service* and *Placement Engine* as subcomponents. The *Infrastructure Controller* is in close collaboration with management applications such as an *Infrastructure Optimizer*. The other 3 hosts each run a *Host Controller*. The dashed line shows the control path from the *Infrastructure Controller* to all of the *Host Controllers*. After requests have been sent to create virtual routers, servers, and virtual links, we observe a virtual topology that spans across these 3 hosts, although any number of topologies can be created across the hosts. The solid black line represents the virtual links.

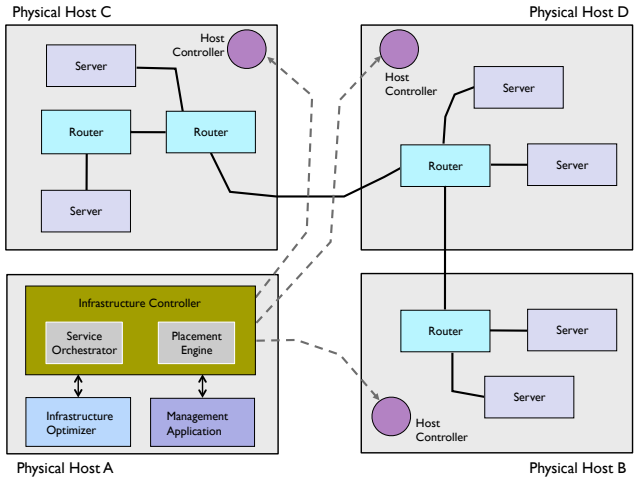

Fig. 2. Mapping and allocation of elements to hosts

B. Lifecycle Management and Control

To start the whole VLSP system, we execute the *Infrastructure Controller* and pass in an XML file which has the configuration options for each particular run, specifying a Java class called: `usr.globalcontroller.GlobalController`. The options include information such as the ports to listen on; the hosts that the *Host Controller* is to execute on and its associated ports; its monitoring elements; the placement engine to use; and other virtual entity options. When the *Infrastructure Controller* starts it attempts to setup all of the monitoring elements and start all of the *Host Controllers* on the specified hosts, this is a Java class called: `usr.localcontroller.LocalController`. Once this has been achieved the VLSP system is ready for experiments.

The interface between all the VLSP components uses HTTP. All communicated information is transmitted using REST and JSON descriptions. To maintain the lightness of the system we use the `monoid Resty` jar library, which is only 120K. This compares to the commonly used, but fully featured, Jersey library which is 1.5Mb. Some example REST calls to the *Infrastructure Controller* from Management Applications are shown in Table I. There are many more REST calls which are all described in the associated documentation of VLSP, available at [24].

Function	HTTP	Call	Result
Create a router	POST	/router/	Router ID
List all routers	GET	/router/	List of Router IDs
Get router info	GET	/router/r	Router info for ID r
Destroy a router	DELETE	/router/r	Confirm router r is gone
Create a link	POST	/link/?router1=r&router2=s	Link ID
List all links	GET	/link/	List of Link IDs
Get link info	GET	/link/i	Link info for ID i
Delete a link	DELETE	/link/i	Confirm link i is gone

TABLE I. EXAMPLE INFRASTRUCTURE CONTROLLER REST CALLS

The start up and shutdown of virtual routers is managed by the *Infrastructure Controller* but is performed by the *Host*

Controller which resides on each host after receiving REST calls. The *Host Controller* is also used to control the connection of virtual routers with virtual links. The *Host Controller* behaves in the same way a hypervisor does in other virtualised environments, and it also passes on *Infrastructure Controller* commands to Routers. A virtual entity will be started on the same physical machine as the *Host Controller*.

C. Monitoring

The monitoring software used in VLSP is called *Lattice* and has been used for monitoring virtualised services in federated cloud environments [25], for monitoring virtual networks [26], and as the monitoring system for an information management platform that aggregates, filters, and collects data in a scalable manner within virtual networks [27] [16]. Lattice [22], which is also an open-source software, has been proven to be ideal for the task of collecting monitoring data for various types of dynamic network environments.

Each virtual router has a set of probes which generate data for virtual link usage, cpu / memory usage per thread, and virtual applications resource consumption. This data is sent to the *Monitoring Engine* of the *Orchestration Layer* which processes it. This data is used by the *Placement Engine* for determining where a new virtual router is placed.

D. Virtual Entity Networking

The virtual entity component (e.g. router or network function) is implemented in Java. The routers have virtual network connections to the other virtual routers they it directly connected to, and exchange routing tables to determine the shortest path to all other routers. A system of sockets and ports is exposed with a *DatagramSocket* interface very similar to standard "sockets". Virtual applications can be run on the virtual routers, thus acting as virtual servers or VNFs, and these can listen to and send data on their associated virtual sockets. Data packets are sent between routers and queued at input and output. Datagrams have headers with a source address, destination address, protocol, source port, destination port, length, checksum and TTL. Many of the features of real IP packets are replicated in the virtual domain. As such, we can take existing Java software that runs on real hosts, and run it on the virtual routers, with a small effort required for porting in order to make the Socket code conform our virtual DatagramSocket interface.

Virtual routers in the system send all traffic, including routing tables and other control messages, via the virtual network interface. Our implementation supports both UDP and TCP communication with virtual and physical network interfaces.

E. Probabilistic Experiment Control

Many experiments that are undertaken in the networking domain are based on arrival rates that are taken from a probability distribution. Within VLSP we have an event engine that can generate events to create a router, destroy a router, create a link, and delete a link, all based on probability distributions. For any particular run we can specify these distributions together with their associated parameters in an XML configuration file. The options for the distribution are set in a `Type` field as one of: Uniform, Exponential, LogNormal, or PoissonPlus, matching well-known topology models. Furthermore, we are providing pre-defined configuration files with realistic topologies, e.g. for data centers.

F. Dynamic Programmable Control

A dynamic programmable control environment allows the definition of scenarios, resources, or other software entity parameters using appropriate functions. In our case, we use the Clojure language [28] for the dynamic configurations. This allows us to perform fully *Software-Defined Operations* using a functional language that has an expressive representation of the configuration settings, while being very brief.

In the following example, we define a linear topology with n nodes and their associated links:

```
;;; Create linear topology
(defn topo-line
  "Create a linear topology with a given name and size"
  [name_val size]
  (let [topo-name (if (symbol? name_val)
                      name_val
                      (symbol name_val)) ]

    (if (< size 2)
      (throw (Exception. "Size too small. Minimum size 2."))
      (do
        (topo-create-fn topo-name)
        ;; Create r0
        (topo-add-router-fn topo-name (str topo-name "-" "r0"))

        (loop [i 1]
          (when (< i size)
            (let [r-name (str topo-name "-" "r" i)
                  prev-r-name (str topo-name "-" "r" (- i 1))]
              ;; create router r-i
              (topo-add-router-fn topo-name r-name)
              ;; create link r-i
              (topo-add-link-fn topo-name prev-r-name r-name)
              (recur (inc i))   ;; i will be incremented
              )))
          (eval topo-name) ) ) ) )
```

Calling `(topo-line linear-example 10)` defines a network structure of 10 nodes and 9 links. This structure is only a *software representation* of the virtual network. We have another function the *activates* the network onto the infrastructure, and hence deploys the virtual routers and virtual links. We use Clojure for networks, for services, and also for static configuration files.

G. VLSP Visualization

One of the management tools we have built for VLSP is a Network Visualization tool. It takes a logical graph view of the virtual topology, including the routers and the links, and presents a visualization graph by using the `graphviz` tool. From data held by the *Orchestration Controller*, a version of the network is generated in the `dot` language that `graphviz` uses. As `dot` is very flexibile, it is easy to create extensions to the visualizations which include the virtual applications that execute on the routers. Furthermore, we are able to present key nodes in different shapes and colours in order to highlight the different features of a topology. As an example, consider the topology in Figure 3 which represents a partial snapshot from one of the experiments in [29]. Those with a similar colour are

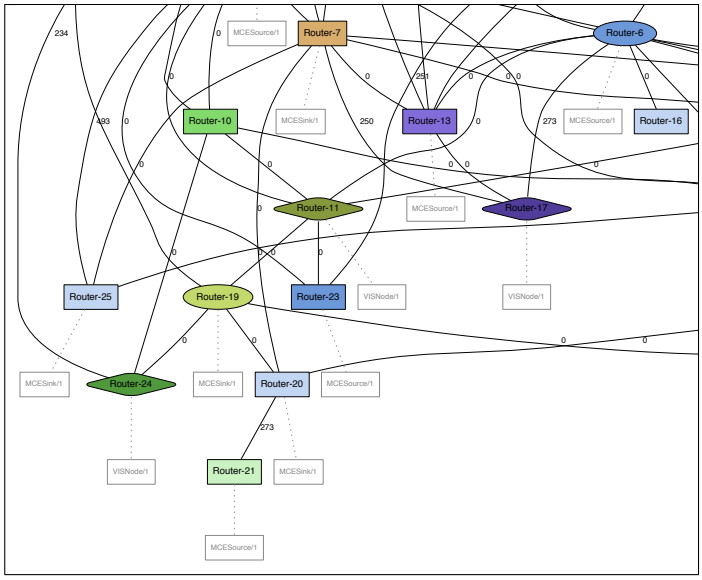

Fig. 3. Snapshot from Network Topology Visualization Tool

logically connected for managing the same data streams, and those nodes represented as a diamond shape are running a Data Proxy application. It allows us to see how data management applications are deployed across the whole topology.

Now the VLSP environment has been described, experimental results derived from test runs undertaken on the implemented infrastructure will be presented.

III. Experimentation Scenarios

In this section we present our evaluation and validation of VLSP for managing virtual entities. Some of this has been presented in our paper [29], where we present two evaluation scenarios in detail, whereas this paper has a more general overview of the experimentation.

A. Deployment

Here we show: (i) the scalability, the flexibility, and the adaptability of the VLSP for a diverse set of topologies and network service deployments; and (ii) that VLSP can operate with a wide range of network and service environments. In all of our tests we used the following hardware: (i) 2 servers with 2 Intel 2.5GHz CPUs (4 cores) and 8GB of memory, (ii) 4 servers with 8 AMD Opteron 2.347GHz CPUs (4 cores) and 32GB of memory, and (iii) 5 servers with 16 Intel Xeon 2.27GhZ CPUs (4 cores) and 32GB of memory.

We deployed various topologies across a set of servers in our testbed, showing the rapid creation and removal of topologies that can enable dynamic services. Our goal is to highlight the following management aspects: efficient resource utilization, dynamic resource elasticity, SDI lifecycle automation, and service placement automation. We see the speed of creating virtual routers and virtual links, as well as their deletion times. We measure the lightweightness of VLSP through observing the router / link startup and deletion times for creating a number of representative virtual topologies. We created: (i) a 10 X 10 grid; (ii) a tree with a spanout of 4 and depth of 4; and (iii) a tree with a spanout of 16 and depth of 2.

We programmatically created the topology descriptions using the Clojure language. The function `(topo-grid 10 10)` creates a 10 X 10 Grid, the function `(topo-tree 4 4)` creates a Tree (4, 4), and the function `(topo-tree 16 2)` creates a Tree (16, 2). Any size grid, tree, or any other topology can be devised and created by VLSP.

To demonstrate the distributed nature of VLSP which operates over a number of physical servers, we deployed each of the 3 topologies across all of our machines in the physical test-bed. After 10 iterations each time, we gradually decreased the number of servers by one, under the control of the *Orchestration Layer*. The experiment stops when the number of physical servers is too small to accept the required number of virtual routers. In table II, we show the number of virtual routers and virtual links in each of the three topologies.

Topology	No of routers	No of links
Grid (10 X 10)	100	188
Tree (4, 4)	85	84
Tree (16, 2)	17	16

TABLE II. Topology sizes w.r.t router and link numbers

Each experimental run starts with the creation of a new network topology. During this time, we gathered the following metrics for our analysis:

Router Startup Time - the time taken to start a virtual router. This time includes the JVM creation time, the loading of the relevant classes, and the time to initiate the required elements to ensure the router is in a *ready state*.

Link Startup Time - the time taken to start a virtual link between two virtual routers. This process includes the negotiation between the routers to set up both ends of the link and ensuring it is in a *ready state*.

After the topology is created, it is shut down to gather the deletion times.

Router Deletion Time - the time taken to delete a virtual router. Deletion includes stopping all executing virtual applications and shutting down all the virtual links attached to that particular router. Deleting a link also requires negotiating with the other end of the link to ensure its disconnection and consistent state. Finally the JVM exits.

Across all of the tests, our average measurements were: (i) router creation around 800ms, (ii) link creation time around 180ms, and (iii) router (plus link) deletion times around 30ms. The test runs have been executed 10 times to ensure replicability of our observations. We saw that 10 replications produced a very low standard deviation of the values.

B. Experimentation

VLSP has been used in many experimental situations that have benefited from its flexibility, adaptability, lightweightness, and scalability.

We have experimented with various placement algorithms, more details of which can be found in our works [30], [31] and [27]. The *Placement Engine* of VLSP is the management component in charge of performing the actual placement of the virtual entities and the application nodes they host, according to the initial topology and the resource usage of the virtual network elements. This is an important feature because, when

we configure a network or a service, given initial context information, some of these parameters may change during the course of the system's operation and a reconfiguration may be required to maintain optimized behaviour. Consequently, our approach has a mechanism to achieve adaptation in a flexible manner. The decision of the *Placement Engine*, which can be changed at run-time under software control, is encoded in an algorithm which can be either rather simple, such as counting the number of virtual routers on a host, or it can be complex, based on a set of constraints and policies that represent the network properties.

A management application we also designed and implemented is the Virtual Infrastructure Information Service (VIS) [32]. VIS realizes logically-centralized information handling for management applications, including optimization of information exchange using the orchestration facilities of VLSP. In general, any logically-centralized service or management facility can be built on top of these abstractions. All the facilities are distributed, but have the advantages of centralized systems as well, i.e. the global view and holistic operation. A *Flow Controller* optimizes the deployed data flows and aligns their configuration parameters (such as routing policies) to the global performance goals in the system and the locally expressed requirements from the data sources and data clients. The latter components set their requirements at the data flow establishment phase. The *Flow Controller* balances the local and global requirements using a negotiation heuristic that considers the existing conditions in the network environment as well. We have carried out a number of data flow optimization experiments using VLSP, described in [23].

The VLSP has been used to implement an Information-Centric Networking (ICN) testbed. Specifically, it has been utilized in the implementation of the CURLING ICN algorithm [33], which employs a hop-by-hop hierarchical content-based publish-subscribe paradigm to content distribution. It is the foundation of a proof-of-concept implementation of CURLING (also known as the coupled approach) of the EU FP7 COMET project [34] In CURLING, two main entities are specified, namely the Content Resolution and Mediation Entity (CRME) located at the control plane which is responsible for resolving content request and the Content-Aware Forwarding Entity (CAFE) located at the data plane which is responsible for the correct delivery of the content towards content clients. In the platform, they are implemented as applications on top of VLSP's virtual routers. These entities enforce the content resolution protocol based on the provider route forwarding rule over VLSP including one of the main ICN components, the in-network caching capability.

Within the DOLFIN project [35], the aim is to optimize the energy consumption within the confines of a single Data Center and also across a group of Data Centers, based on system virtualization techniques and the optimal distribution and placement of virtual machines. To realize such a system, we created a *software Data Center* using a VLSP topology mapped over many physical servers. We then augmented the VLSP with energy-aware monitoring, energy-aware resource allocation, coupled with energy modeling, which gave us the ability to model the amount of energy being used by a virtual element. These extentions where integrated with a Smart Grid system called Open ADR, which dispatches energy prices to consumers. Using this system we were able to model the actual cost of energy per Virtual Machine, cost of energy per host, and cost of energy per DC.

IV. Conclusions

We introduced the Very Lightweight Software-Driven Network and Services Platform (VLSP), which is specially designed to meet the characteristics of managing flexible and dynamic environments such as SDIs. It has integrated network management and control operations for important software defined elements (i.e. virtual routers, servers and network functions) through a logically centralized element. We have evaluated VLSP and shown that: (i) it is lightweight and suitable for scalability and stability tests, (ii) it is an implementation allowing the evaluation of diverse scenarios.

VLSP achieved better simplicity and non-functional characteristics over using a hypervisor running a standard virtual machine and standard OS (i.e. improved scalability; lower resource utilization; quicker startup speed; reduced heaviness; eliminate the issue where most of the router functionality is not needed; and more networking flexibility). Finally, VLSP is fully distributed and modular. This has been achieved, as the management components as well as the host controllers and the virtual entities themselves can be deployed across any number of physical hosts interacting via REST calls.

From the experimental usage of VLSP we have determined:

What is VLSP good for?

- Testing and evaluating alternative SDN / NFV scenarios
- Mid scale tests of network software written in Java (100s and likely 1000s of virtual routers).
- Testing software robustness to "unexpected" network conditions (sudden "rude" start up/shut down exposes software deficiencies).
- Testing software robustness to unreliable networks.
- A compromise between simulation (realism questionable) and large testbed (requires many physical machines).
- Comparing simulation with testbed results

What is VLSP not *yet good for?*

- It is not optimized for forwarding performance, compared to a real router it routes packets at a slower speed and uses more overhead (i.e. due to the focus on the support of new network management and control features).
- It is difficult to support facilities and protocols that rely on maximum bandwidth calculations, e.g. traffic engineering algorithms estimating the link bandwidth.
- The direct interaction with or the driving of hardware interfaces is out not currently addressed.

Finally, we plan to explore the usage of VLSP for flexible service creation within the context of Software-Defined Infrastructures, especially focusing on network service chaining [36], [37]. This involves extensions of the VLSP with methods and processes for continuous dynamic operation of services, service composition, and aggregation of service blocks, including service delivery based on orchestration, programmability and automatic (re)deployment [38]. Further extensions include augmenting the internal routing functions with flow table traffic forwarding mechanisms to allow SDN style management.

ACKNOWLEDGEMENT

This work was partially supported by the EU projects: DOLFIN [35], 5GEx – "5G Multi-Domain Exchange" [39] and SONATA – "Service Programing and Orchestration for Virtualized Software Networks" [40].

REFERENCES

[1] OpenStack, "OpenStack Open Source Cloud Computing Software," Tech. Rep., http://www.openstack.org.

[2] OpenDaylight, "SDN and NFV platform that enables network control and programmability," Tech. Rep., http://www.opendaylight.org.

[3] OPNFV, "Open Platform for NFV," Tech. Rep., http://www.opnfv.org.

[4] OOR, "Open overlay router," http://openoverlayrouter.org/.

[5] A. Rodriguez-Natal, M. Portoles-Comeras, V. Ermagan, D. Lewis, D. Farinacci, F. Maino, and A. Cabellos-Aparicio, "Lisp: a southbound sdn protocol?" *Communications Magazine, IEEE*, vol. 53, no. 7, pp. 201–207, July 2015.

[6] Telefonica, "Open mano," https://github.com/nfvlabs/openmano.

[7] ETSI ISG, "Network functions virtualisation (nfv) architectural framework." [Online]. Available: {http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf}

[8] ONF, "Software-Defined Networking: The New Norm for Networks," ONF White Paper, Tech. Rep.

[9] ETSI, "ETSI Network Functions Virtualization," Tech. Rep., http://www.etsi.org/technologies-clusters/technologies/nfv.

[10] M. P. Anastasopoulos, A. Tzanakaki, G. S. Zervas, B. R. Rofoee, R. Nejabati, D. Simeonidou, G. Landi, N. Ciulli, J. F. Riera, and J. A. Garcia-Espin, "Convergence of heterogeneous network and it infrastructures in support of fixed and mobile cloud services," in *Future Network and Mobile Summit*. IEEE, 2013, pp. 1–9.

[11] UNIFY, "Unify project," https://www.fp7-unify.eu/.

[12] T-Nova, "T-nova project," http://www.t-nova.eu/.

[13] Neutron, "OpenStack Networking - Neutron," Tech. Rep., https://wiki.openstack.org/wiki/Neutron.

[14] R. Morris, E. Kohler, J. Jannotti, and M. F. Kaashoek, "The click modular router," in *ACM Transactions on Computer Systems*. Citeseer, 2000.

[15] Quagga, "Quagga Routing Suite," http://www.quagga.net.

[16] L. Mamatas, S. Clayman, M. Charalambides, A. Galis, and G. Pavlou, "Towards an information management overlay for emerging networks," in *Network Operations and Management Symposium (NOMS)*. IEEE, 2010, pp. 527–534.

[17] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," *SIGPLAN Not.*, vol. 48, no. 4, pp. 461–472, Mar. 2013.

[18] A. Kivity, D. Laor, G. Costa, P. Enberg, N. HarÉl, D. Marti, and V. Zolotarov, "Osv – optimizing the operating system for virtual machines," in *2014 usenix annual technical conference (usenix atc 14)*, vol. 1. USENIX Association, 2014, pp. 61–72.

[19] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 459–473.

[20] A. Madhavapeddy and D. J. Scott, "Unikernels: Rise of the virtual library operating system," *Queue*, vol. 11, no. 11, pp. 30:30–30:44, Dec. 2013.

[21] A. Madhavapeddy, T. Leonard, M. Skjegstad, T. Gazagnaire, D. Sheets, D. Scott, R. Mortier, A. Chaudhry, B. Singh, J. Ludlam *et al.*, "Jitsu: Just-in-time summoning of unikernels," in *12th USENIX Symposium on Networked System Design and Implementation*, 2015.

[22] S. Clayman, "The lattice monitoring framework open-source software," http://clayfour.ee.ucl.ac.uk/lattice/.

[23] L. Mamatas, S. Clayman, and A. Galis, "A service-aware virtualized software-defined infrastructure," *Communications Magazine, IEEE*, vol. 53, no. 4, pp. 166–174, 2015.

[24] S. Clayman, L. Mamatas, and A. Galis, "The very lightweight software-driven network and services platform (vlsp) open source software," University College London, http://clayfour.ee.ucl.ac.uk/usr/.

[25] S. Clayman, G. Toffetti, A. Galis, and C. Chapman, "Monitoring services in a federated cloud-the reservoir experience," *Achieving Federated and Self-Manageable Cloud Infrastructures, IGI Global*, 2012.

[26] S. Clayman, A. Galis, and L. Mamatas, "Monitoring virtual networks with lattice," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*. IEEE, 2010, pp. 239–246.

[27] S. Clayman, R. Clegg, L. Mamatas, G. Pavlou, and A. Galis, "Monitoring, aggregation and filtering for efficient management of virtual networks," in *Proceedings of the 7th International Conference on Network and Services Management*. International Federation for Information Processing, 2011, pp. 234–240.

[28] Rich Hickey, "The clojure programming language," 2008, http://clojure.org/.

[29] S. Clayman, L. Mamatas, and A. Galis, "Efficient management solutions for software-defined infrastructures," in *IFIP/IEEE NOMS 5G Man*, 2016.

[30] S. Clayman, E. Maini, A. Galis, A. Manzalini, and N. Mazzocca, "The dynamic placement of virtual network functions," in *1st IEEE / IFIP International Workshop on SDN Management and Orchestration*, 2014.

[31] R. G. Clegg, S. Clayman, G. Pavlou, L. Mamatas, and A. Galis, "On the selection of management/monitoring nodes in highly dynamic networks," *Computers, IEEE Transactions on*, vol. 62, no. 6, pp. 1207–1220, 2013.

[32] L. Mamatas, S. Clayman, and A. Galis, "A flexible information service for virtualized software-defined network," *Submitted to International Journal of Network Management (Special Issue on Software-Defined Operations) on 31/8/2015*, 2015.

[33] W. K. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. G. De Blas, F. J. Ramon-Salguero, L. Liang, S. Spirou, A. Beben *et al.*, "Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services," *Communications Magazine, IEEE*, vol. 49, no. 3, pp. 112–120, 2011.

[34] A. Beben, "The content mediator architecture for content-aware networks," pp. 1192–1203, 2012.

[35] DOLFIN, "Data centres optimization for energy-efficient and environmentally friendly internet (dolfin fp7 project)," http://www.dolfin-fp7.eu.

[36] N. Yadav, J. Guichard, B. McConnell, C. Jacquenet, M. Smith, A. Chauhan, M. Boucadair, P. Quinn, R. Manur, P. Agarwal *et al.*, "Network service chaining problem statement," *Network*, 2013.

[37] A. Manzalini, R. Saracco, C. Buyukkoc, P. Chemouil, S. Kuklinski, A. Gladisch, M. Fukui, E. Dekel, D. Soldani, M. Ulema, W. Cerroni, G. Schembra, V. Riccobene, C. Machuca, A. Galis, and J. Mueller, "Software-Defined Networks for Future Networks and Services," White paper based on the IEEE Workshop SDN4NS 2013, Tech. Rep.

[38] A. Galis, S. Denazis, C. Brou, and C. Klein, *Programmable networks for IP service deployment*. Artech House, 2004.

[39] 5GEX, "EU H2020 - 5G Multi-Domain Exchange (5GEX) project," https://5g-ppp.eu/5gex/.

[40] SONATA, "EU H2020 - 5G Service Programing and Orchestration for Virtualized Software Networks (SONATA) project," https://5g-ppp.eu/sonata/.