Virtualized ICN (vICN): Towards a Unified Network Virtualization Framework for ICN Experimentation

Mauro Sardara, Luca Muscariello, Jordan Augé, Marcel Enguehard,
Alberto Compagno, Giovanna Carofiglio
Cisco Systems, firstname.lastname@cisco.com

ABSTRACT

The capability to carry out large scale experimentation and tests in real operational networks is crucial, to assess feasibility and potential for deployment of new networking paradigms such as ICN. Various platforms have been developed by the research community to support design and evaluation of specific aspects of ICN architecture. Most of them provide ICN-dedicated, small scale or application-specific environments and ad-hoc testing tools, not re-usable in other contexts nor in real-world IP deployments.

The goal of this paper is to contribute *vICN (virtualized ICN)*, a unified open-source framework for network configuration and management that leverages recent progresses in resource isolation and virtualization techniques to offer a single, flexible and scalable platform to serve different purposes, ranging from reproducible large-scale research experimentation, to demonstrations with emulated and/or physical devices and network resources and to real deployments of ICN in existing IP networks.

In the paper, we describe vICN rationale and components, highlighting programmability, scalability and reliability as its core principles, illustrated by means of concrete examples.

1 INTRODUCTION

The encouraging results of ICN research efforts in the last years have triggered broader industrial interest in ICN as a serious candidate to relieve future 5G network challenges in terms of performance, scalability and cost (see e.g. latest ITU recommendation developed by the Study Group 13 [7] and 5G Americas White Paper on ICN and MEC [13]).

To bridge the gap between a promising network architecture and a feasible deployment-ready solution, to introduce in existing operating network infrastructures, experimentation at scale and in real-world environments is a critical step. From that, it depends the ability to show clear benefits over the state of the art and to convince about the feasibility of the integration of the technology into existing network infrastructure.

Information-Centric Networking (ICN) community has largely stated the importance of a pragmatic application-driven and experimental research approach from the beginning (see e.g. [24],[16]). Multiple tools and testbeds have been developed for simulation and emulation (CCNx, NDN software and testbed, CCNlite, MiniCCNx [3], MiniNDN). Most of existing tools have been designed to serve the purpose of assisting research and specifically design and evaluation of aspects of ICN architecture (e.g. caching, forwarding or routing). To this aim, they operate in dedicated fully-ICN network environments, trading-off abstraction of network characteristics for scale and offering limited flexibility to modify core ICN features, network topology and settings, application APIs.

In this paper, we aim at complementing existing tools with *vICN (virtualized ICN)*, a flexible unified framework for ICN network configuration, management and control able to satisfy a number of important deployment and experimentation use cases: (i) conducting large-scale and fine-controlled experiments over generic testbeds; (ii) instantiate reliable ICN network supporting real applications in proof of concepts; (iii) the ability to deploy large networks within network for trial and test development.
Clearly, requirements are different: research experimentation needs fine-grained control and monitoring of the network and reproducibility of the experiments. Prototypes for demonstration require a high level of programmability and flexibility to combine emulated and real network components or traffic sources. More than in previous cases, reliability and resource isolation is a critical property for deployments in ISP networks.

The operations required for the deployment of an ICN network include to install/configure/monitor a new network stack in forwarder elements and at the end-points and the socket API used by applications. If loading a network stack from an application store into general purpose hardware is easy to realize, a network stack has different requirements compared to a cloud based micro-service: ultra reliability, high-speed and predictability, to cite a few. vICN shares the same high level goals as SDN/NFV architectures, but with additional ICN-specific capabilities not typically required by IP services. Overall, we identify three main challenges that vICN addresses and that differentiate it w.r.t. state of the art:
(i) *Programmability*, i.e. the need to expose a simple and unified API, easy to facilitate bootstrap, expressive enough to accommodate both resource configuration and monitoring and flexible enough to allow the user to decide about level of control granularity. Existing softwares like OpenStack, which are built as a collection of independent components, each one following different design patterns, do not offer a satisfactory level of programmability.
(ii) *Scalability*: vICN software infrastructure aims at combining high-speed packet processing, network slicing and virtualization, highly parallel and latency minimal task scheduling. Current systems are based on a layered architecture that does not permit overall optimization, so limiting scalability on the long term.
(iii) *Reliability*: a fundamental property of vICN which consists in maintaining the state of deployment, recovering from failures and performing automatic troubleshooting. This requires the overall software to be able to accommodate programmable function monitoring and debugging. In existing designs, each component has independent implementations to achieve that.

The remainder of the paper is organized as it follows. Sec.2 summarizes the state of the art, before introducing vICN architecture in Sec.3 and its implementation in Sec.4. Sec.5 provide concrete examples of vICN in action. Sec.6 concludes the paper.

2 RELATED WORK

Salsano et al. [18] proposed to introduce network virtualization for ICN networks through the use of Openflow. In [15], the authors address a similar issue and propose an architecture to perform network slicing. However it misses fundamental aspects of network management and control. Mininet [9] makes a step in that direction and sits closer to our objectives as it allows the creation of virtual networks based on containers and virtual switches. It allows to test application performance in emulated network conditions by allowing setting parameters such as link delays and capacity, node CPU share, etc. However, it does not propose any slicing mechanism, and lack support for wireless or any control on applications or workload.

The Cloud computing community has made important efforts to facilitate the use of datacenter resources. Cloud Operating Systems have been proposed, such as OpenStack [5], designed to manage and monitor large-scale deployments, providing access to network, compute and storage resources through a set of homogeneous APIs and sub-projects. Available tools are generally oriented towards applications being deployed in a global pool of resources. Container-specific tools such as Kubernetes [12] present some interesting aspects in that they expose a unique consistent API for simplicity, with however limited control granularity our purpose. Automation is ensured by third party tools layered on top of these standard APIs, like in Chef [22] which are intrinsically limited by their procedural language design, which requires to explicit every step of the deployment, manually adapt to the current state and explicit error handling. Other tools (e.g., Puppet [11]) use a descriptive language, where the user only needs to describe his/her needs and leave the rest to the tool. Despite the integration effort in Cloud computing, the silos around functionalities and the proliferation of APIs appear limiting for our purpose. The system is not suitable for allowing simple setup and control from users, nor to build applications on top.

The now joint SDN and NFV communities are maybe the closest to our needs, at least from an architectural point of view. In [6] for instance, the authors describe a set of design principles for the Management and Orchestration (MANO) of virtualized network functions (VNF). Their approach is based on a modular architecture, clearly identifying the fundamental function such as user and VNF description, orchestration, etc. They also point the need to ensure reliable management by considering the lifecycle of the resource they manage. OpenDayLight is a promising candidate framework for building NFV capabilities, as it relies on a model-driven abstraction layer which fits with our requirements. However, this aspect is mainly used from a software engineering point of view, and not to offer programmability of the resource, or micro-services as they are referred to. Moreover orchestration is layered on top of other modules which behave as silos.

3 THE VICN FRAMEWORK

The high-level architecture of vICN, presented in fig. 1 reminds of NFV proposals such as MANO. Our contribution is in applying the underlying resource end-to-end and globally, and the properties this guarantees.

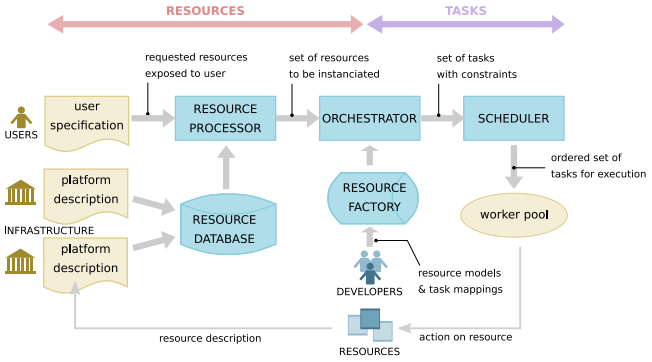

Figure 1: vICN functional architecture

3.1 Functional architecture

The resource is the basic unit of information in vICN, that can be combined and extended to form other resources. It consists in an abstract model that a set of mappers use to translate actions on the resource into a series of task to be executed. vICN architecture differentiates the role of users, of developers and of infrastructure providers. Developers enrich the tool with a set of resources – stored in the **resource factory** – which can be very diverse, ranging from a specific Virtual Machine to an application such as a DNS resolver or an IP route. Both users and infrastructure providers use this base set of resources to describe what they respectively require (users) or make available (infrastructure providers).

Resources are specified according to different degrees of detail: e.g. the infrastructure is described very precisely, to form a **resource database** which will serve as a base for deployment. Instead, user specifications might the quite general or abstract on purpose, and only mention resources of interest for the user. The role of the **resource processor** is to turn an abstract and incomplete description into a set of resources mapped on the infrastructure leading to a consistent deployment.

Once selected the resources, the **orchestrator** translates them into a set of actions to be performed, based on the current state of deployment regularly monitored and on constraints due to task synchronization or sequentiality. The resulting actions are processed by a **scheduler** which outputs an execution plan and dispatches parallel tasks to a worker pool with the objective of minimizing the deployment time.

To summarize, vICN is based on two main abstractions: *resources* which are external unit of information exposed to users, developers and infrastructure providers, and *tasks* which are internal units of information, defined by developers, to translate resource requests into changes of network deployment state.

3.2 Resource model

In vICN, the internal resource model is exposed directly to users and developers through a query language, in the spirit of SQL or SPARQL, that builds on and extend an *object relational model* [4].

The model defines a base *object* as a set of typed attributes and methods, where types refer to standard integers, strings, etc, or to newly defined object themselves. The query language built on top

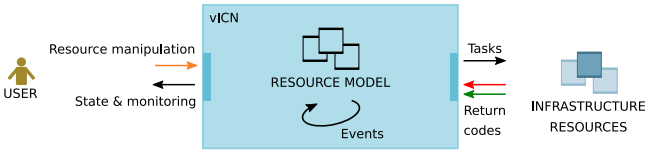

Figure 2: Flow of information in vICN

is used to create, destroy and manipulate those objects, either for resource setup or to retrieve monitoring information.

This model benefits from the well-recognized power and expressiveness of the relational algebra [14] and of some key concepts of Object-Oriented Programming, namely composition and inheritance. This allows to build an integrated interface based on both human- and machine-readable semantics (as in YANG [1]).

Resources. Resources in vICN are logical representations of physical and/or remote elements whose state has to be kept synchronized. The state of a resource can be affected by user queries, by events involving resources, or through monitoring queries issued to the remote resource. Indeed, events are used to implement loose coupling between resources: e.g. a routing component might recompute routes every time it is informed about a change in the set of nodes, interfaces or links. The flow of information is represented in fig. 2.

Resource are also annotated by the tool with their state and a queue of pending changes.

Resource state. The state of a resource is tracked, for reliability and consistency, by a Finite State Machine (FSM) presented in the left part of fig. 3. The FSM presents the possible states of a resource (rectangles, raising events) and the pending operations, or tasks, being executed (round shapes). The transitions are dictated by user's actions (or internal events) and follow the typical lifecycle of an object:
INITIALIZE is called when the shadow resource and its object are being created for internal setup; *CREATE* and *DELETE* are the respective constructor and destructor: they can create or destroy the remote resource and eventually set some attributes; *GET* retrieves the current state of a resource, as well as the state of some of its attributes; *UPDATE* proceeds to attribute update, and in fact runs parallel instances of attribute-FSM, as showing on the right-hand side of the figure.

Resource mapper. A developer can associate commands to be executed for each transition between states by means of resource mappers. These commands are handled by vICN through the task abstraction, which also inherits from the base object, and are specialized to cope with multiple southbound interfaces such as NETCONF/YANG, SSH/Bash or LXD REST calls (this architecture reminds Object-Relational Mappers such as SQLAlchemy [2]).

New tasks can be created through inheritance or composition, using algebraic operators to inform about their parallel or sequential execution. Resource objects are equipped with similar operators, so that inheritance and composition produce a similar composition of tasks. Both resources and tasks define an algebra, and the scheduler will be able to use this property to perform calculations and optimize the execute plan.

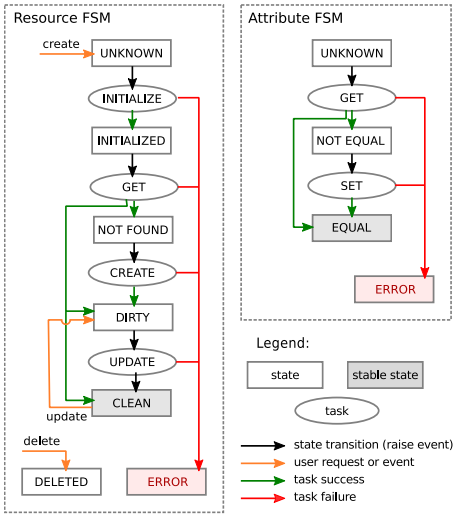

Figure 3: vICN Finite State Machine

A subset of resources defined in vICN is represented in fig. 4, showing in particular the four base abstractions of Node, Interface, Channel and Application from which most resources inherit, and which reminds of those defined in [10] for instance.

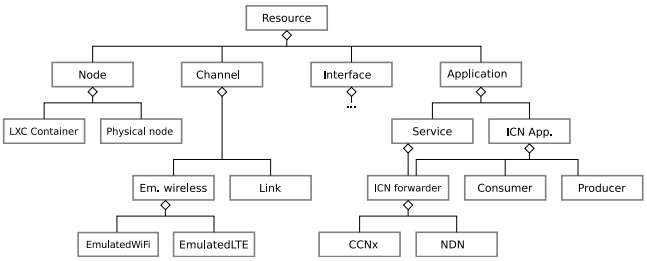

Figure 4: vICN partial Resource Hierarchy

3.3 Resource processor

As previously described, the resource processor must adapt the user specification to available resources. It does this through: (i) *specialization* of abstract resources (e.g., a Node) into concrete ones (e.g. a Linux Container) thanks to type inheritance; (ii) *mapping* of resources to infrastructure (e.g. deciding on which server a container will be hosted) thanks to a placement algorithm; and (iii) *inferring* missing or incomplete resource specifications according to the resource model semantics and to rules specified by the developers. (e.g. a container will require a LXD hypervisor and a bridge for connectivity).

Such assignment can be assimilated to an (NP-Hard) Constraint-Satisfaction Problem (CSP) [8] as the system has to accommodate several resources in a finite capacity system in terms of networking, compute and memory. Both user-specified attributes and optional platform policies are taken into account in the CSP as additional constraints. The output is a mapping from specification to implementation, which will also be used to expose back monitoring to the user in a consistent way.

3.4 Orchestrator and Scheduler

The role of the orchestrator is to maintain one FSM per resource, and ensure they reach the desired state requested by the user. Its outcome is a task dependency graph which is shared with the scheduler. Task dependencies are derived from resources dependencies, structure of the FSM, as well as from inheritance and composition constraints related to both the resources and the mappers.

The scheduler ensures the scalability of the deployment by scheduling the parallel execution of tasks over a pool of worker threads. Given the dependency graph presented before, this corresponds to a classical DAG scheduling problem, which has been widely studied in the community [19]. Figure 5 presents a toy-scenario underlining the need for not naive scheduling algorithm. In that small example, a greedy selection of the task with higher distance to destination is a sufficient heuristic to get an optimal solution and save one execution round. We remark that because of user interactions, the graph of tasks might evolve in time and require a recomputation. In that sense, heuristics [17] might be preferred to optimal solutions because of their faster execution time, while providing satisfactory performance.

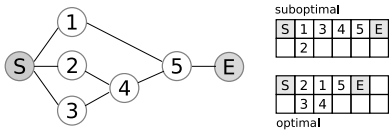

Figure 5: Toy scenario - vICN scheduler

Because of its centralized architecture, vICN performance is also impacted by the network transmission time[1]. We alleviate this issue by enabling task batching, when two consecutive tasks target the same node interface. The algebraic structure of tasks also makes it possible to reorganize the graph structure to better optimize execution, or to increase the ability to batch tasks.

4 IMPLEMENTATION

The flexibility of vICN lies in its modular architecture organized around its resource model. This allows various resources to be developed in order to fit with a wide range of underlying infrastructure, and bring missing functionalities such as slicing or topology management to cite a few. We further describe the current release and a set of base resources covering the whole ICN stack. They build on and reuse available technologies that we have selected with scalability and reliability in mind.

4.1 vICN codebase

A first version of the vICN code base has been open-sourced within the Community ICN (CICN) project [20], as part of the Linux Foundation's Fast Data I/O effort. The code, written in Python, is released under the Apache v2.0 software license. This software implements all the building blocks described in fig. 1 and is mature enough to launch complex ICN deployments.

Along with it, we distribute a prepackaged ICN image containing the full CICN suite (including the ICN forwarders, stack, and useful applications), so that a full ICN network can be bootstrapped in a matter of tens of seconds or minutes. This suite includes a high speed forwarder based on the VPP framework Fast Data I/O framework [23] which can already offer almost 1Mpps per thread in its first release.

[1] network round-trip-time and, for instance with TLS, session establishment time

4.2 Slicing

In addition to bare-metal deployments, vICN allows to slice nodes and links offered by the infrastructure through a set of technologies we describe here. This is crucial for proper experiment isolation, and to realize a separate control and management plane.

Virtual nodes can be implemented as both containers or virtual machines. Containers have been preferred as the core technology (via the use of LXD) because they are more lightweight and efficient (thanks to zero-copy mechanisms, ZFS filesystem and simplified access to the physical resources), despite increased security concerns and limitations such as sharing the same kernel. This should not be limiting since most ICN functions are implemented in userland, while drastically increasing performance.

Network is shared at layer 2 via OpenVSwitch [21], which provides advanced functionalities, such as VLAN and OpenFlow rules, required by our wireless emulators and to bridge external real devices to the virtual environment. vICN fully isolates the deployment's network from the outside world by creating a single and isolated bridge per deployment, using iptables as a NAT to provide external connectivity. On top of that, we reduce the load of the bridge and isolate control traffic from the data plane. Indeed, we directly link connected containers, through pairs of Virtual Ethernet interfaces (veth), thus bypassing the bridge. Connected containers that are spawned across different compute units in a cluster are transparently connected through a GRE tunnel.

Finally, vICN has to arbitrate for shared resources on the physical host, be it container or interface names (with constraints such as the 16-character limit on Linux), VLAN IDs, and even MAC or IP address depending on the level of required network isolation. It is important to do such "naming" properly not only for correctness, but also to simplify debugging and troubleshooting. vICN further enforces consistent names that uniquely identify a resource, which allow for faster detection and recovery when the tool restarts or has to redeploy the same experiment.

4.3 IP and ICN topologies

Using the mechanisms described previously, it is possible to build a layer-2 graph on top of which vICN can set up both IP and ICN topologies.

For IP networking, a centralized *IP Allocation* resource is in charge of allocating subnets or IP addresses to the different network segments of the graph. Global IP connectivity is then ensured by computing the routes to be installed on the nodes. vICN provides a generic *routing module* implementing various algorithms (such as Dijkstra or Maximum-Flow) taking as an input a graph (layer-2) and a set of prefix origins (allocated IP addresses), and outputting a set of routes, which are themselves resources. Route setup is driven by a *Routing Table* resource, from which the Linux Routing Table or the VPP one (i.e. Vector Packet Processing [23]) inherit.

The process is similar for ICN, except that we first build (IP or Ethernet) faces based on a configurable heuristic (e.g. L2 adjacency). We can then reuse the same Routing Component by feeding it with the face graph, and the set of prefix origins found in attributes of content producer resources. The corresponding Routing Table is, in this case, implemented by the ICN forwarder. We remark that the use of multipath routing schemes (e.g. Maximum Flow) make more sense in this context.

The process results in a deployment accommodating IP and ICN coexistence, allowing performance comparison of both architectures at the same time.

4.4 Link emulation

A feature that is missing from most tools is the ability to measure the performance of applications running on top of virtual networks with specific bandwidth or propagation delays. vICN offers resource attributes interfacing with the Linux Traffic Control layer (tc) in order to effectively shape link bandwidth and emulate constrained networks.

A complementary aspect is the ability to use emulated radio resources as an alternative to real hardware in a transparent fashion. Two types of radio channel are currently supported, WiFi and LTE, both based on real-time simulation features of the NS-3 simulator. The vICN radio channel resource is implemented as a drop-in replacement of a regular radio resource connecting stations and access point (or UEs and Base Station) through a configurable radio channel, and hides the internal wiring from the user.

The emulation is then taking care of all relevant wireless features such as *beaconing, radio frequency interference, channel contention, rate adaptation and mobility*. We scale real-time emulation by using multiple instances orchestrated by an overarching mobility management resource in vICN, communicating in real time with the different emulators. This process can collect relevant information from the simulation, and expose it to the internal model and thus monitoring.

4.5 Monitoring capabilities

Monitoring is natively implemented as part of vICN as a transversal functionality, building on the object model introduced in Section 3. The query language offered by vICN allows to query any object attribute, including annotations made by the resource processor and orchestrator about the host or the deployment state of the resource. This is the same interface that is used by the orchestrator to query the current state of a remote resource, to communicate with the emulators, or for the user to interact with vICN in order to change an attribute or create a new resource at runtime.

Its syntax closely matches SQL syntax, more precisely a query object containing the following elements: the object name, a query type (create, get..), a set of filters and attributes, eventually completed by attribute values to be set.

For periodic measurements such as link utilization, vICN provides a daemon that can be installed on the nodes and exposes information via a similar interface. Communication between the components is ensured using the IP underlay setup by vICN.

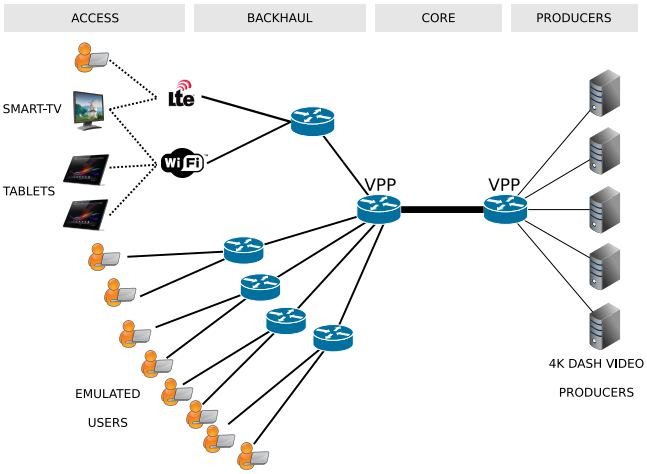

Figure 6: Mobile World Congress topology

5 EXAMPLES

We now illustrate some characteristics of vICN using a particular use case: mobile video delivery. This section is not meant to be exhaustive, but to illustrate how the design of vICN helped us solve practical challenges, and to emphasize general properties of the design that are relevant to other use-cases.

5.1 Use case description

The recent years have seen drastic changes in the video consumption patterns that put much pressure on delivery networks: the shifts in video quality (up to 4K), from broadcast to on-demand and from fixed to wireless and mobile networks. Our objective was to show that ICN addresses these challenges, using mechanisms like caching or multihoming over heterogeneous networks. Figure 6 represents an example of such a video delivery network. It consists of four parts: an heterogeneous WiFi/LTE access network with multihomed video clients; a backhaul network aggregating the resulting traffic with workload from emulated clients; a core network composed of two nodes; and producers serving 4K video. All nodes have a fully-featured ICN-stack. The core nodes use a VPP-based high-speed forwarder, the others a socket-based one. Overall, the deployment consists of 22 LXC containers, 3 real devices connected to the virtual network, 22 emulated links (including WiFi and LTE channels), and one physical link between DPDK-enabled network cards (the core). We use a pre-packaged container image containing all the necessary software to reduce the bootstrap time.

5.2 Scalability

The simplification offered by vICN is illustrated the following numbers: during the deployment, vICN created about 800 resources compared to the 104 declared in the topology file, a reduction in complexity of 85-90%. More than 1500 bash commands were executed, either directly on physical machines, or on LXC containers. This even underestimates the number of Bash commands an operator would type to deploy an equivalent topology, as some are batched for efficiency reasons (e.g., we insert all IP routes for a given node in a single command).

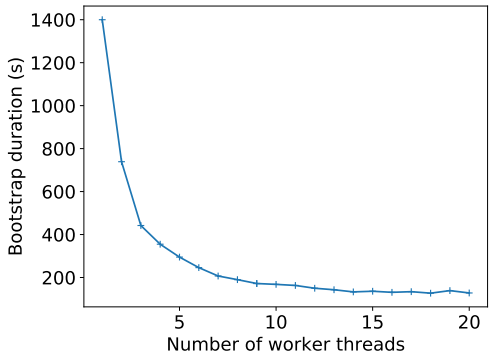

Figure 7: vICN bootstrap time as a function of the number of worker threads

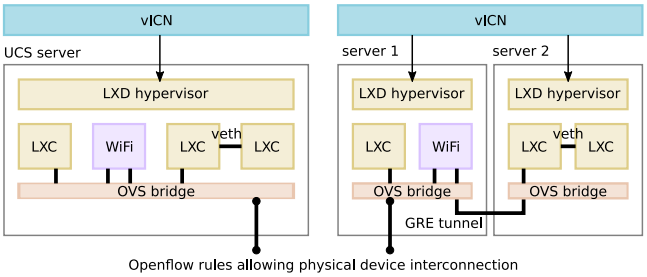

Figure 8: Alternative vICN topology deployments on single server and a cluster.

Figure 7 then shows the time taken by vICN to deploy the topology as a function of the number of dedicated threads. We deployed this topology on a Cisco UCS-C with 72 cores clocked at 2.1 Ghz. We first note that multi-threading provides a sevenfold reduction in bootstrap time, and that the topology can be deployed in about two minutes. The observed gains are due to the I/O-intensive nature of tasks, which spend most of their lifetime waiting for return values. This reduction is specific to our implementation and our simplistic scheduling heuristic. The shape of the curve remains nonetheless interesting, with a performance bound appearing. This is due to the underlying task graph, whose breadth intrinsically limits the number of tasks that can be run in parallel.

5.3 Programmability

One advantage of our resource model (see section 3.2) is the use of inheritance. It allows the user to chose his level of granularity depending on his or her needs and expertise. In particular, the user can remain oblivious to the underlying technology used to deploy resources. We used that feature to scale the demonstration on a cluster of servers connected through a switch instead of a single powerful server. In that configuration, linking containers on different hosts requires to connect them to virtual bridges on their respective hosts, and to link these bridges through a L2-tunnel. The two deployments, shown in fig. 8, require different resources and tasks. However they can be realized with the same vICN specifications, thanks to the Link abstract resource. Here, vICN completely abstracts the implementation complexity and enables painless switching from one deployment to the other.

The deployment of containers running VPP is another example of vICN's ability to shield a user from implementation and configuration details thanks to its resource model. Indeed, VPP uses DMA access to contiguous memory areas named hugepages. Both the host and the containers have to be configured to allocate and share enough of these hugepages. On top of starting and setting up the application on the container, VPP thus requires to execute commands on the physical node and to change the container's configuration before its creation. In vICN, simply linking VPP to a container is enough to perform the bootstrap. The tool is then able to change the other resources (e.g., use a VPP-enabled container instead of the standard one) and to run all the necessary commands.

The flexibility of the framework also allowed us to switch resources in many occasions. During our tests, we replaced real tablets by emulated nodes to generate test workloads. During the demonstrations, we could also seamlessly use a real LTE mobile core instead of an emulated one. It only required to change one resource in the specification and did not affect the rest of the scenario.

5.4 Monitoring and Reliability

We conclude by highlighting how the resource model enables monitoring and debugging. As described in section 4.5, vICN exposes a query language based on its underlying model for monitoring. This language can be used to collect information about the network status at different time scales: link utilization, radio status, cache status etc. vICN thus integrates all information about the deployment in a consistent and query-able representation, building on the model introduced in section 3. In the same way vICN provide an API to navigate through structured logs which may assist the whole process of software development.

6 CONCLUSION

ICN community has developed multiple tools for simulation and emulation to assist design and experimentation. In this paper, we introduce *vICN (virtualized ICN)*, a flexible unified framework for ICN network configuration, management and control to complement existing tools, especially for large scale and operational networks deployment. vICN is an object-oriented programming framework rooted in recent advances in SDN/NFV research that provides higher flexibility that existing virtualization solutions and is specifically tailored to ICN. While most of current software is developed in silos, with significant limitations in terms of optimization, vICN offers the capability to optimize each component of the virtual network to provide carrier-grade service guarantees in terms of programmability, scalability and reliability.

vICN design, that we illustrate briefly through a concrete example, comes with a free software implementation available in the Linux Foundation for the community with multiple objectives: demonstrations, research and field trials. We leave for future work the detailed presentation of vICN characteristics by means of benchmarking in such different use cases.

REFERENCES

[1] Martin Bjorklund. 2010. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020. (Oct 2010). DOI: https://doi.org/10.17487/rfc6020

[2] Amy Brown. 2012. *The architecture of open source applications (SQLAlchemy)*. Vol. 2. Kristian Hermansen.

[3] Carlos Cabral, Christian Esteve Rothenberg, and Maurício Ferreira Magalhães. 2013. Mini-CCNx: Fast prototyping for named data networking. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 33–34.

[4] Christopher John Date and Hugh Darwen. 1998. *Foundation for object/relational databases: the third manifesto*. Addison-Wesley Professional.

[5] The OpenStack Foundation. 2017. https://www.openstack.org/. (2017).

[6] European Telecommunications Standards Institute. 2014. *Network Functions Virtualisation (NFV); Management and Orchestration*. Technical Report GS NFV-MAN 001. European Telecommunications Standards Institute (ETSI).

[7] ITU. March 2017. Recommendation ITU-T Y.3071 Data Aware Networking (Information Centric Networking): Requirements and Capabilities. In *ITU Study Group 13 Final Report*. https://www.itu.int/rec/T-REC-Y.3071-201703-P

[8] Alan K Mackworth. 1992. Constraint satisfaction problems. *Encyclopedia of AI* 285 (1992), 293.

[9] Mininet. 2017. http://mininet.org/. (2017).

[10] NS3. 2017. The Network Simulator 3. https://www.nsnam.org/. (2017).

[11] Puppet OpenStack. 2017. https://wiki.openstack.org/wiki/Puppet. (2017).

[12] Production-Grade Container Orchestration. 2017. https://kubernetes.io/. (2017).

[13] 5G Americas White Paper. December 2016. Understanding Information-Centric Networking and Mobile Edge Computing. ... http://www.5gamericas.org/files/3414/8173/2353/Understanding_Information_Centric_Networking_and_Mobile_Edge_Computing.pdf

[14] J. Paredaens. 1978. On the expressive power of the relational algebra. *Inform. Process. Lett.* 7, 2 (1978), 107 – 111. DOI: https://doi.org/10.1016/0020-0190(78)90055-8

[15] Ravishankar Ravindran, Asit Chakraborti, Syed Obaid Amin, Aytac Azgin, and Guoqiang Wang. 2016. 5G-ICN : Delivering ICN Services over 5G using Network Slicing. (2016). http://arxiv.org/abs/1610.01182

[16] Future Internet Research and Experimentation. 2017. https://www.ict-fire.eu. (2017).

[17] Rizos Sakellariou and Henan Zhao. 2004. A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*. IEEE, IEEE, 111.

[18] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri. 2013. Information Centric Networking over SDN and OpenFlow: Architectural Aspects and Experiments on the OFELIA Testbed. *Comput. Netw.* 57, 16 (Nov 2013), 3207–3221.

[19] Oliver Sinnen. 2007. *Task scheduling for parallel systems*. Vol. 60. John Wiley & Sons.

[20] The Linux Foundation. 2017. Fast Data project (fd.io) Community ICN (CICN). http://wiki.fd.io/cicn. (2017).

[21] The Linux Foundation. 2017. Open vSwitch. http://openvswitch.org/. (2017).

[22] The Linux Foundation. 2017. OpenStack Chef. https://wiki.openstack.org/wiki/Chef. (2017).

[23] The Linux Foundation. 2017. Vector Packet Processing - Fast Data I/O. https://wiki.fd.io/view/VPP/. (2017).

[24] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *SIGCOMM Comput. Commun. Rev.* 44, 3 (Jul 2014), 66–73. DOI: https://doi.org/10.1145/2656877.2656887