Challenges for Managing the Disappearing Network – beyond SDN

Sven van der Meer*, Eduard Grasa†, Bernat Gastón†, Micheal Crotty‡, Jason Barron‡,
Victor Alvarez Roig§, Marc Sune§ and John Day¶
*Ericsson NM Labs, Ericsson Software Campus, Athlone, Co. Westmeath, Ireland, email: sven.van.der.meer@ericsson.com
†Fundacio i2cat, Barcelona, Spain, email: eduard.grasa@i2cat.net, bernat.gaston@i2cat.net
‡Telecommunications Software & Systems Group, Waterford Institute of Technology, Ireland
Email: jbarron@tssg.org, mcrotty@tssg.org
§BISDN GmbH, Berlin, Germany, Email: victor.alvarez@bisdn.de, marc.sune@bisdn.de
¶Boston University, Boston, MA

***Abstract*—Software Defined Networks (SDN) has taken the world by storm. Only a few years old as technology, most of the big players have SDN in their product portfolio or in their strategic roadmap. SDN has changed the way we virtualize the network fabric in data centers, provided new features for cloud computing, and arguably plays a big role in facilitating Network Function Virtualization (NFV). Looking ahead, SDN has the potential to make the network disappear altogether, similar to Mark Weiser's vision for computing. However, while SDN's main contribution is a new south-bound interface for TCP/IP flow control, little work has been done on the north-bound interface for the interaction with and the management of an SDN network. Two essential items are still missing. First, SDN does currently not provide means to expose network capabilities to applications (e.g. a QoS cube), thus it fails to bridge the gap between the network and services. Second, SDN does not help to advance network management while introducing many new challenges for it. In this paper, we start discussing the disappearing network and discuss how to address the two missing items in the progress. Our contribution is to examin the Recursive InterNetworking Architecture (RINA) as an evolutionary step for SDN, which we present that in the form of three use cases.**

I. Introduction

The idea of Software Defined Networks (SDN) originated at Stanford University and was first described in [1] in 2008. The past four years have seen SDN breaking into data centers and telecommunication carrier networks. A recent study by Infonetics Research [2] shows two main factors that drive the adoption of SDN: (a) increased service agility that allows for quicker time to revenue and (b) a global view of a multi-vendor, multi-layer network with a standardized interface. Proof of concept deployments have been seen in 2013, larger trials are being moved from labs into the field in 2014 and the first commercial deployments are expected to come in 2015 [2]. So, only a few years old as technology, most of the big players have SDN in their product portfolio (e.g. Ericsson [3]) or in their strategic roadmap (e.g. AT&T [4]).

SDN is that successful because it has done to networks what cloud computing not that long ago has done for computing and storage. First, it decoupled software from specialized hardware by virtualizing the underlying physical resource. Second it offered a (semi-) standardized interface to the hardware (and the remaining software artifacts running on it, e.g. applications and components). And third it came with (relatively) easy yet real-world bounded deployment scenarios. Cloud computing, e.g. OpenStack, has shown the added value of virtualization and SDN has taken full advantage of that applying it to network resources.

Being a very young technology, and in terms of real-world deployments very immature as well, it might be very early to ask what the future for and with SDN will be. However, we believe that SDN could do for networks what Ubiquitous Computing has done for computing. To paraphrase Mark Weiser [5]: the network will disappear; become part of everyday life. This means that today's Software Defined Networks facilitate tomorrow's Software Defined Networking. With networks becoming a fully programmable resource, we can not only change switching and routing behavior on existing protocols. Instead we can introduce any kind of new protocol - and subsequently any kind of new protocol stack/suite - inject it into the network and let it coexist with any other kind of network stack/suite (with all consequences this will have).

This means that we can now, for real and in the field, define very specialized and optimized (virtual) networks. The fifth generation of mobile networks (5G, e.g. [6]) for example will provide infrastructure for a wide and diverse service mix: tiny bandwidth and low QoS for sensor networks, low and high bandwidth with low latency for isochronous types of services (low bandwidth for voice, medium to high bandwidth for video), and varying bandwidth and medium latency for web and email access. Larger developments (such as the Internet of Things) and industrial visions (such as Ericsson's Networked Society [7]) are driving the service mix.

What is SDN missing today to be a truly disruptive technology and to make the network disappear? In short: a North-Bound Interface (NBI). In more detail: two very important items (as discussed in [8]). First, SDN will need to allow applications to configure the network according to their requirements. Instead of exposing all available information, an interface should allow applications to express the Quality of Service (QoS) they require, thus allowing the network to optimize operations for it. Second, SDN will need to provide for a consistent and coherent approach to manage an SDN network. Little has been done to provide for complimentary management functions that can deal with the very new situation that SDN creates.

In this paper, we are first discussing the state of play in the wider SDN world (section 2). Based on our prediction that SDN can make the network disappear, we then discuss what that means and introduce one approach on how it can be done. This approach is the Recursive InterNetworking Architecture (RINA, described in section 3). Current RINA implementations can be used to adopted already on top of an SDN controller or as a substitution for traditional network stacks. Following that, we discuss how new management functions will need to be defined (section 4) along with three use cases (section 5) and explain how that will help to harmonize networks and applications/services (section 6).

II. State of Play in SDN

As Lara states in [9]: "OpenFlow is currently the most commonly deployed SDN technology". The main objective is to separate switch control software from the actual switch hardware. This allows deploying and testing new switching behavior in real networks without changing the underlying hardware. Standardizing the interface between the (new separate) control software (control layer) and the data processing software on the switch (data layer) provided the basis for wider exploitation and created industrial interest. For a comprehensive discussion on OpenFlow, including the underlying protocol and deployments, see [9].

Looking at SDN beyond Openflow, we can see approaches that offer new ways to realize programmable networks. Nunes et al. discusses a number of present and future advantages for SDN-based programmable networks in [10] (e.g. deployment scenarios and network control granularity). This survery also looks into the related research challenges. First, controller and switch design will need to be adapted. Second, the aspect of SDN-based internetworking comes into play when SDNs in different administrative domains need to be connected. Thirs, the interaction between the controller and services remains an open issue today.

The two surveys [9] and [10] clearly demonstrate the amount of progress that has been made for the southbound interface, i.e. the communication between the SDN controller and an SDN-enabled switch or router. A few projects introduce mechanisms to coordinate multiple controllers (e.g. as a transparent proxy [11]). The NBI, i.e. the functionality exposed by an SDN controller and the applications (or management systems) using it, is largely ignored.

One exception is the work of Kim and Feamster [12], which addresses two problems: the continuously changing networks state (which change much faster in programmable networks compared to today's physical ones) and the required low-level, per device network configuration. Kim and Feamster developed an innovative policy language called Procera. This language is based on Functional Reactive Programming (FRP). Operators can define rather complex policy in a declarative way. Procera's execution environment, embedded in Haskell, then executes the policies and enforces the dynamic network changes. The event-based networking module of Nick Feamster's SDN course [13] demonstrates the capabilities of this approach.

Some larger projects emerge from the European research program. One example is the Unify project [14]. Unify started in 2014 and aims to integrate cloud and networking with two main objectives. First, they build a joint orchestration for cloud and network. Second they provide means to invoke dynamic service chains on top of the orchestration. This involves a Network Function Forwarding Graph (NF-FG) similar to ETSI's Forwarding Graph (FG) in the Network Function Virtualization (NFV) standards [15]. A discsussion of research challenges for managing SDN and VNF in a telecommunication environment, with special focus on real-time aspects, can be found in [16].

III. There is No Network

Software defined networks and their interface OpenFlow are changing the way how we build and use networks. Those networks become by and large a virtualized resource. SDN not only solve many problems in today's networking but also expose new ones. Interestingly, it also offers a way out of those problems at the same time. A network with SDN becomes a truly distributed software system, while some of the original design criteria of networks do not change. So while the application model changed, the underlying network did not (OpenFlow does flow control only):

- We have devices that can execute multiple applications (the device is no longer equivalent to the application, as it was in telephony). The network should be routing data between applications, but it continues to route data between interfaces of devices: the network still thinks that interconnecting physical interfaces is the same than interconnecting applications
- Different applications have different requirements, but the network, by and large, still provides a single level of service (best effort for IP networks).
- There is no way for applications to communicate their requirements to the network, other than requesting a reliable (TCP) or unreliable (UDP) flow.
- There is no way for the network to understand the applications' requirements (and vice versa) unless an external system (in telecoms for instance a BSS/OSS) mediates between them.
- As there is no notion of application names in the network, applications have no way of knowing what remote applications are requesting a communication to them. Therefore, access control rules can only be specified with combinations of IP addresses and port numbers, making them very expensive to maintain: every time the mapping of an application to IP address and port changes - renumbering, mobility - the access control rules have to be manually updated.

With virtualization (for compute, storage and now network resources), those problems need to be confronted. It is no longer sufficient that the network only route packets to the physical interface of the computer, and port numbers. We now need to also differentiate between applications executing in the "host OS" and "virtual machines", since each VM can use the full range of TCP/UDP ports.

Now consider the network stacks we find in real deployments today. Take a simple data center with a physical network (Ethernet) connecting a number of switches, some routers and a lot of blades. On the blades we have VM's in some cloud environment, maybe OpenStack. The resulting networks stacks do not follow the "transport/network/data link/physical" architectural model. We can find L2 over L2 (MAC-in-MAC), L2 over L3 (Ethernet over MPLS), L2 over L4 (VXLAN, L2TP), L3 over L3 (IP in IP, GRE, LISP), and so on. The L1/L2/L3/L4 model does not match the reality.

What are the problem and current SDN solutions showing us? In short: routing is not feasible in the long run. We need to change towards routing between applications, on all levels of the soon completely virtualized world. And, as demonstrated above, the current architectural model does not exist anymore. In long we can see three main problems:

- A fixed network stack of layers doing different functions does not work, because it assumes that all the security and resource allocation problems are solved just once, in a particular way in the entire network (there is a single scope). As one would expect and reality is showing, this does not scale. The number of scopes in the network is not fixed and varies on different parts of the network, therefore there can be no limits in the number of layers defined a priori by the architecture if we want it to scale.
- Each layer in the stack can be modeled as a distributed application (what is usually referred to as an overlay) that uses the layer below to communicate with the "nearest neighbors". TCP and UDP are distributed applications that use IP to communicate. IP is also a distributed application that uses different data link layers to communicate, and so on. If we allow no limits in how these overlays may be combined (except that L1 is always directly on top of the physical media), we can recreate the structure we see in practice. But this model is still complex and chaotic: there are different types of layers, without well-defined interfaces and no rules on how to compose them.
- If each layer is a distributed application, we see that the problem requires a recursive model: each layer at the N-level in the stack requires that the layer below (N-1) connects together the protocol machines at the N-layer. Therefore for the N-1 layer the N-layer protocol machines are applications, and the N-1 layer has to move data between these applications, effectively providing an InterProcess Communication service (IPC Service). The N-1 layer protocol machines are the applications of the N-2 layer protocol machines, and so on until we reach the physical medium (this is where recursion stops).

This analysis shows that an architecture to better support distributed computing is formed by a number of indefinite layers providing Inter Process Communication (IPC) services to each other, we will call each one of these layers a Distributed IPC Facility or DIF. Each DIF provides a well defined interface that allows applications on top to communicate with each other by name, specifying the characteristics required for each service (bandwidth, delay, jitter, reliable delivery, in-order delivery, etc). DIFs provide services to each other down to the physical media, wrapped by a shim DIF. A shim DIF is the equivalent of the physical layer in the current computing model, and just hides the particularities of a certain physical media behind the DIF interface. This is the architecture model provided by RINA [17].

Further exploring the above analysis we can now make a number of statements. Exploiting the full potential of SDN and programmable networks means that we can not only control flows but also control the network as a whole, for instance the deployed protocol stacks. That means that we can deploy several different protocol stacks that coexist. Now, network management is an even harder problem, since all commonality (coherent protocol stack) is gone.

SDN natural evolution leads to an architecture that address the mentioned problems, simplifies the nework and translates the control of the network to the applications. This architecture is RINA, a change of paradigm: network is no nore the problem to provide a requested service, but the solution. *The network then becomes a distributed systems based on Inter-Process Communication, i.e. in Mark Weiser's words it 'disappears' - there is no network.*

In other words, a real SDN á la RINA unifies distributed computing and networking, since networking is simply a distributed application: distributed computing specialized to provide IPC services. Now an application can request communication services to other applications by specifying the application name and the characteristics required for the service, without having to know the internal details of the network (DIF) that is providing this communication service, and without having to know where the destination application is executing. Virtualization is part of the architecture. If virtualization is understood as a mechanism to isolate a security and/or resource allocation domain, than this is exactly what a DIF is. Furthermore, the behavior of each DIF can be customized to optimally adapt to its operating conditions (as shown in the next section). All-in-all, this approach provides a better framework and toolset to support distributed computing.

This approach is backed by underlying theory, which is originally described as "Patterns in Network Architecture" [18]. It was further developed by the Pouzin Society (PSOC) [19] in form of a set of specification, resulting in research activities funded by the NSF and the European Commission. The theory of "networking is inter-process communication (IPC) and only IPC" is detailed in [20]. The European FP7 project IRATI [21] has published an implementation of the RINA architecture. Figure 1 shows the RINA architecture. For detailed dicussions on each part of the architecure please see the RINA education page [22] and the RINA podcast series [23]. For an example of an operator's perspective see [4]).

IV. Managing the disappearing Network

While the network disappears, i.e. becomes a largely a software artifact similar to an operating system, it still needs to be managemed. At a first glance, controlling and managing a programmable and disappearing network is not much different from traditional management and control. The main problems are: routing, congestion control, distributed and efficient resource allocation, resilience, security and reliability.

Looking deeper shows that, by using RINA for SDN, allows for a new look at how we manage the disappearing network as a distributed system. We call the resulting system a Distributed Management System (DMS) to emphasize that we manage multiple collections of IPC processes (i.e. layers).

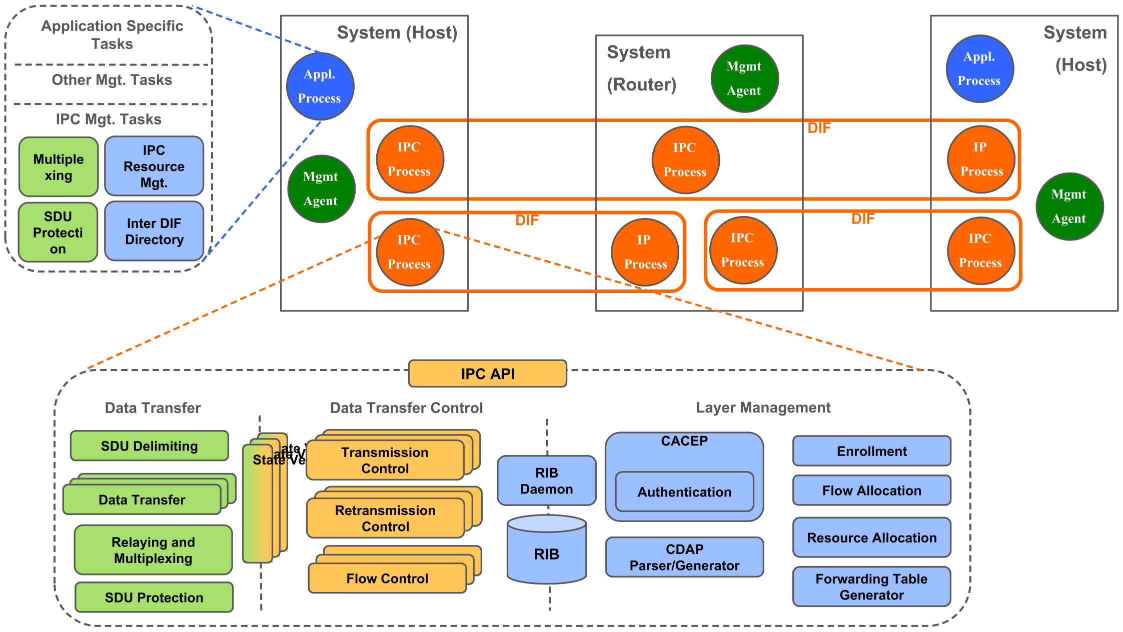


Fig. 1: Recursive InterNetworking Architecture - RINA

1) An application is represented by an Application Process (AP). An AP can use IPC to communicate with other APs. More importantly, an AP is executed somewhere, usually on top of an Operating System (OS) of some form. So the first point of management is this very OS to ensure that its compute, storage and networking resources provide the required services for the APs and their communication. We can call this the OS-DMS.
2) A DIF is a collection of two or more APs cooperating to provide IPC. This is the second point of management: remotely manage one or more elements of a DIF. We call this the NM-DMS, since it resembles the current concepts of network management (including the traditional manager/agent paradigm).
3) A collection of DIFs form a layered network (overall) and a protocol stack (inside a single system). This is the third point of management: manage a set of DIFs as a whole application. We call this the AM-DMS.
4) The last point of management is dedicated to coordinate a given name space. We call it the NSM-DMS. The need for this point of management comes from the new naming architecture, which focuses on applications rather than network interfaces and ports. Each application process and IPC process has a unique name within the AP name space. Within a DIF, each IPC gets a synonym (i.e. address). This synonym is only visible inside the DIF, which means IPCs of the same DIF can communicate with each other. To use a lower layer DIF, a point of attachment is used to be able to address IPC there without knowing their internal synonym. Since this mechanism is recursive, it requires some coordination to maintain uniqueness of addresses inside a process and DIF name space and to maintain the points of attachment.

All elements (processing system, APs, IPC processes) are managed by one DMS, the lowest point of management being the OS-DMS. NM-DMS and AM-DMS can monitor all resources of non-DMS DIFs. We call the collection of all elements managed by a single DMS the DMS Domain. DIFs can be managed by more than one DMS; elements (processes; see above) can only be managed by a single DMS. This requirement ensures that direct process management conflicts can be resolved in the managing DMS regardless of the number of managing DMSs of the DIF the process is contained in. In the extreme case, all elements of DIF are managed by a different DMS, e.g. an OS-DMS.

Do four points of management make the actual management more complex? Not when we consider the system they manage. RINA defines one layer (the DIF, with a complete naming architecture), which is then used multiple times to realize different network functions. All parts of the layer (data transfer, data transfer control, layer management) are identical. This means we can build all required management instrumentations in a general fashion, and then use specific instantiations of them depending on the point of management they are used for. In other words, we are not losing the concept of generic management instrumentation of today's management system while drastically simplifying the actual management tasks. Initially we can use traditional configuration templates and management policies, deploy them, and study how we can advance those to optimize our management system.

The European FP7 project PRISTINE [24], started to build the required management instrumentation in January 2014. This instrumentation and new management concept provides a good framework to investigate the properties (and limitations) of different management strategies (e.g. centralized, decentralized, by delegation, autonomic). In other words, the management of the disappearing network can be adapted to the very specific requirements of many (if not all) different deployment scenarios. This is an important advantage over current, centralized control, SDN solutions.

V. Use Cases

Within the PRISTINE project, we are focusing on three very relevant use cases to understand the manageability of future SDN, i.e. RINA, environments. The first three subsections detail these use cases. The last subsection discusses the advantages of applying RINA to manage the use cases by exploiting the common layer structure.

A. Use Case: Distributed Cloud

The SlapOS (Simple Language for Accounting and Provisioning Operating System) [25] is a decentralized Cloud Computing technology. It is designed to automate the deployment and configuration of applications in a heterogeneous environment, and it relies on servers located in people's home and now also in offices, data centers or even in a smartphone, tablet or TV. One of its main applications is to create a disaster recovery cloud that can resist any force majeure event - which does affect traditional clouds from time to time. In order to provide a high-level of resiliency, security and better manage the distributed resources, the SlapOS cloud service operates on an overlay over the Internet, whose main elements are: a pseudo-random connectivity generation policy, OpenVPN tunnels and the Babel routing protocol [26].

PRISTINE is evaluating RINA as an alternative architecture to build a more scalable, secure, manageable and programmable overlay that better supports the decentralized SlapOS cloud services. Figure 2 (a) illustrates the overlay architecture using RINA. VIFIB nodes are the systems participating in the decentralized cloud, regardless of their location. The SlapOS base DIF is the layer that interconnects all the systems in the cloud, and floats "on top" of the IPv4 Internet or Ethernet LANs (depending on the connectivity available between systems). Service-tree DIFs floating on top of the *SlapOS-base* DIF can be created to further isolate the resources allocated to different users, acting essentially as customizable VPNs where routing, security and data transfer policies can be tailored to different applications.

B. Use Case: Data Center (DC) Networking

DC networks today are moving away from a tiered, hierarchical structure to a structure that resembles more and more that of a supercomputer; where all the nodes can be interconnected amongst them with high bandwidth and minimum delay. However, supercomputers are a far more controlled, static an homogeneous environment than a DC: usually external access is not allowed, there's one single tenant that executes High Performance Computing (HPC) applications on behalf of users and all the systems are at one single location. In contrast, DCs are multi-tenant by nature, must support access from the outside (if providing a public cloud service) and the DC may be actually distributed within multiple locations. Moreover, the support for cloud computing demands flexibility, as applications with different networking requirements are dynamically instantiated and destroyed.

RINA provides a very good framework to design networking solutions for DCs, due to its simple, common but highly customisable architecture. Figure 2 (b) shows the design of a DC network using RINA, whose structure looks very similar to the distributed cloud solution.

Both presented use cases benefit from a common connectivity and resource allocation domain (as provided by the DC Fabric and the *SlapOS-base* DIFs, respectively) that supports layers dedicated to support the needs of different applications/tenants (the *Tenant and Service-Tree DIFs*, respectively). The differences lay on the characteristics of the underlying connectivity media: while in the DC use case all the systems are co-located and the DC owner therefore can completely control resource allocation; in the distributed cloud most systems use the public Internet to connect to each other - constraining performance outputs of the DIFs floating on top.

C. Use Case: Network Service Provider

Network Service Providers are nowadays exploring new possibilities for offering their services in a more secure, resilient and, specially, flexible ways (see AT&T's vision [4]). The concept of NFV [27] decouples network capacity and network functionality in such a way that all functions are software pieces, Virtual Network Functions (VNF), that run on top of regular hardware.

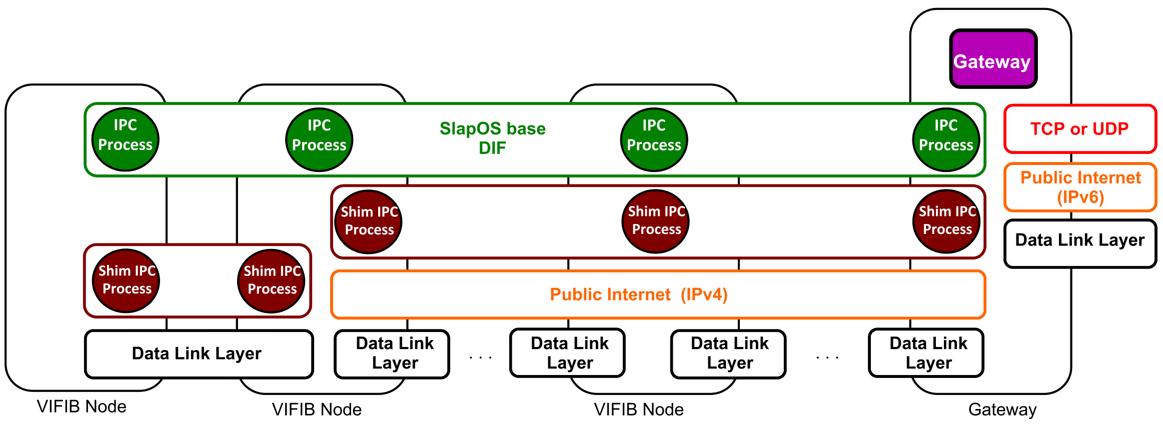
This approach breaks the dependency on specific hardware, and at the same time lets the operator adapt the usage to the real needs at every moment, because the VNFs can be instantiated and removed strictly on demand.

Every VNF is supposed to be very modular. In order to provide more complex services, adapted to the needs of the different users, NFV supports the construction of composed VNFs also known as Service Function Chaining (SFC) [28].

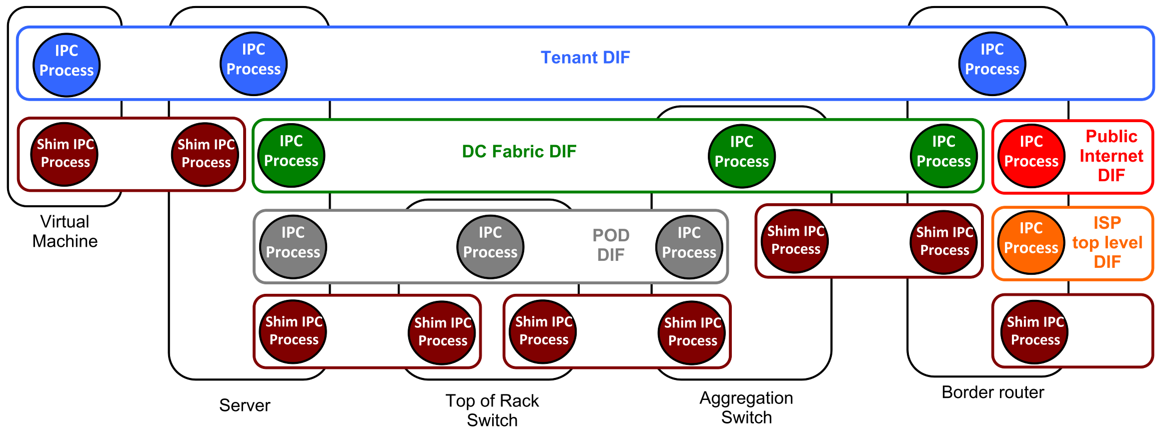
It is worth saying that the RINA architecture is a completely different paradigm with respect to IP networks. RINA paradigm is focused in the concept of IPC and only IPC, and as such, functions like firewalling or load balancing are inherently part of the it. Nevertheless, this use case shows how can IP networks benefit from having a RINA deployment underneath.

Figure 2 (c) shows an example of the different DIFs that could be used to deploy NFVs with RINA. As in the other use cases of PRISTINE, there is the need to have a basic common interconnection DIF which can be used to communicate all the APs. In this case it is called service transport DIF and all the Service Chains DIFs and DAFs lay on top of it. The shim DIFs and service transport DIFS are managed by the provider, whereas each DIF and DAF service chain is configured by its tenant. The VNF1-VNFn Application Processes apply the VNFs to the traffic that they carry. As already mentioned, this traffic is IP, and the functions applied as well, but it is being forwarded by a - completely transparent - RINA network.

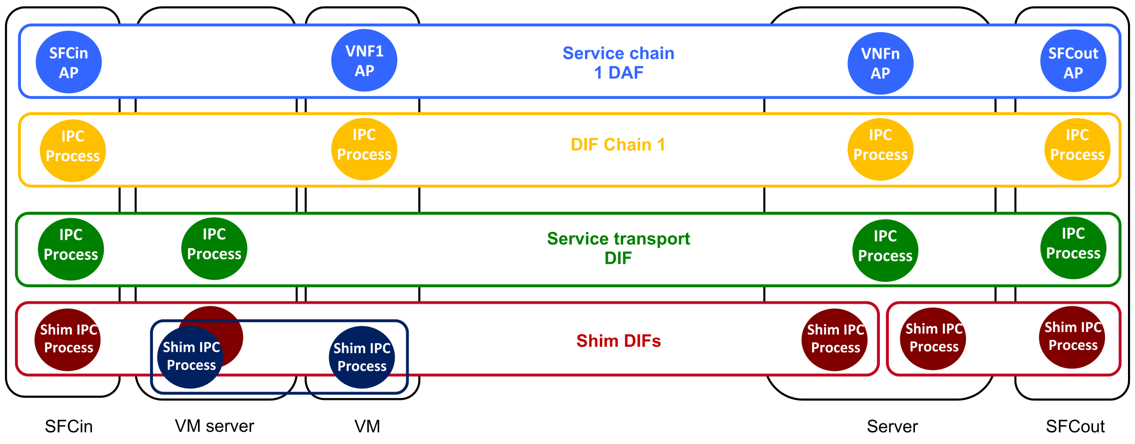
The use of a different Service Chain DAF per tenant ensures the isolation among them, given the fact that each DIF has a different address domain, since a DAF (and DIFs as a degenerate case) are secure containers on its own. This is a significant help in improving the security in a NFV deployment. Another benefit comes by the fact that RINA assumes that all applications are distributed applications, and that gives a major improvement in order to have real distributed service chains.



(a) Distributed Cloud



(b) Data Center Networking



(c) Network Provider: NFV single chain

Fig. 2: PRISTINE Use Cases

D. Advantages of the RINA approach to Management

The common structure of RINA can be exploited by Network Management in a number of ways.

Simplification in the management of multiple layers, yet with the possibility to configure different policies. Today more and more layers are performing functions that were not traditionally assigned to them (such as link-state routing in layer 2 DC networks [29] or in layers implementing the so-called "Virtual Networks"). Layers are transitioning from performing a single function to becoming units of distributed resource allocation, something the RINA model fully predicts. However, since the different traditional layers have evolved independently from different starting points and have been standardized by different committees, the same functions (such as routing, or authentication) have very different semantics.

As a consequence configuration, performance or fault management become complex, since the NMS have to be aware of the details of every specific technology implemented in different layers. In contrast, RINA networks provide the same structure; configuring for example a link-state routing in the "DC-fabric DIF" or in each of the different "Tenant DIFs" with exactly the same semantics - the same objects with different values can be used everywhere.

Dynamic application discovery and DIF creation on the fly via Namespace Management. Name-space management systems allow distributed applications to be discoverable through more than one DIF. Taking the Distributed Cloud as an example, a user on the Internet could request a flow to be allocated to a certain application name, only available via the SlapOS cloud. A distributed Namespace Management system would forward the query until it reached the node in the VIFIB system where the application was running. If the application accepted the flow, the NMS of the SlapOS cloud and the NMS of the user (the OS in the simplest case) could collaborate to create a new DIF that floated over the Internet and the *SlapOS-base* DIF and allowed the user to have private connectivity to the application instance(s).

More effective complex network event management. The commonality provided the RINA architecture enables the definition of a performance model that is common to all layers. That would be a very powerful tool for the Network Management System, since: i) the NMS would be receiving less types of events, freeing it from translating to an internal performance model to reach a proper understanding of the overall network status; ii) the correlation of events in multiple layers would become much simpler, facilitating the identification of problems by the NMS and allowing it to roll-up actions to effectively solve and mitigate them.

More automation is possible: management should be monitor and repair - not control. With RINA the focus of the NMS can shift from "protocols" to "policies", requiring less human intervention from human operators - and hence less human errors. Since protocols would become a commodity - just part of the common, standard layer infrastructure - research and experimentation could shift towards understanding the behavior of different policy-sets under different operating conditions. This would enable network administrators to manage the network the following way:

1) At design time, characterize the different conditions that the network will experiment during its day-to-day operation (e.g. in terms of offered load, failure of links, identified ongoing security attacks, etc.)
2) At design-time, group these conditions into well-defined "operational regions", and choose policy-sets that are effective for each of the different regions.
3) At run-time, let the NMS monitor the network, decide if the measurements belong to one of the "designed" operational regions. If the operational region didn't change, continue monitoring. If the operational region transitioned into another "designed" operational region, automatically apply the related policy-set. If the operational region is unknown - as it could be the case with natural catastrophes, terrorist attacks, etc. - notify the human operator

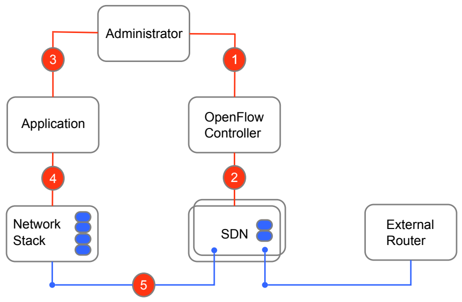


Fig. 3: Existing deployment workflow

VI. Harmonizing the Network with Applications and Services

As discussed earlier, different business sectors are looking at SDN as a solution to increase service agility (quicker time to revenue) in a multi-vendor, multi-layer network (see for instance [2]). Most modern applications - for example real-time streaming multimedia, Voice over IP (VoIP), Voice over LTE (VoLTE) and safety-critical applications - have specific requirements for bandwidth, latency, loss, jitter and error probability on QoS level and very strict demands on Quality of Experience (QoE) level when offering services to users. SDN, today, has no notion of QoS (as one of the two problems introduced before). QoS can be offered by prioritising certain flows based on the requested characteristics (e.g. send best effort through a different path rather than the path used for higher priority ones). QoE can then be used as an offering to users (on service level) that can be mapped to a configuration of the network (QoS, on network level). OpenFlow standardises a network protocol for configuring network switches on-the-fly. However, currently this is an explicit process that requires intervention by system administrators, as shown in Figure 3.

An administrator needs to update the OpenFlow controller to allow a new applications data-flows. First, system administrators need to configure the network policies (1). Second, the controller then configures the SDN switches (2). Third the application needs to be deployed and configured (3). Finally application opens network flows (5) as needed and these result in packets being sent to the relevant SDN switches, which route the packets as per policies.

This approach relies mainly on the coordination between the OpenFlow policies (and potentially many of those polices), and the configured application (IP addresses, port numbers, etc.). Relying on well known transport port numbers is not a scalable solution, thus arbitrary mappings to the underlying data-centre network substrate are not obvious and requires very complex configurations to be maintained.

A possible solution for simplifying this problem, can be found in the RINA architecture [18]. RINA standardises the API for applications and this API includes QoS aspects. Everything in RINA looks like IPC. The repeating structure of layers in RINA, together with the capability of each layer

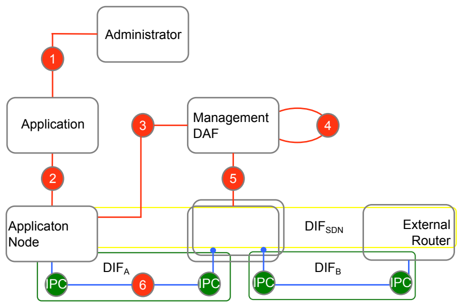


Fig. 4: Required new deployment workflow

to be configured to support different levels of QoS for each individual flow, makes RINA a logical solution to provide multiple QoS services.

Some QoS information should be included in requests from applications so that an automatic network flow mapping can be achieved (see Figure 4).

The application is configured as before(1). First, the application requests network flows with additional QoS attributes (2), Second, the IPC manager decides if an appropriate DIF exists and can support the QoS. If there is an available DIF, a new flow is requested over it, and we jump to step (6). If no suitable DIF is available, the application stack forwards this request on the management DAF (3). Third, the Management DAF selects the applicable policies (based on the information available) (4) and creates a new DIF(5). Finally, traffic can flow between the application and its clients (6).

This allows higher-level policies to automate the network configuration, including forwarding table entry generation and mapping to lower layers in the stack. For example, to encapsulate web streaming data within the appropriate VLAN and sub-network range (to ensure latency constraints).

From the applications point of view, there is a significant decrease in the level of complexity as the tenant only sees a QoS-aware SDN layer. By attaching QoS to the flows, automatic configuration of the lower level network stack is possible. Their view is simplified to application instances and client nodes, yet they still have per flow QoS control over bandwidth and latency. Ultimately, this makes the network disappear, as it looks like just arranging an IPC between two application processes, the management stack takes care of automating the mapping, based on pre-agreed policies.

VII. Conclusion

In this paper we look at Software Defined Networks as one of the most disruptive technologies in the recent past. Discussed first in 2008 as an academic project with a campus test-bed by Stanford University, it has taken the world by storm. We discussed that a lot of work has contributed to SDN by academia and industry, building on its initial success. SDN has changed the way we define the networking of clouds and data centers, it has opened the way for new approaches such as NFV, and it has the potential to change the way we do networking as a whole.

We have also motivated a very likely future of SDN comparing this new technology to developments we have witnessed in computing. Our argument is that SDN can do to networks what Mark Weiser has predicted for computing (and which has very much happened already): it can make the network disappear altogether. SDN changes the network from being a hardware dominated resource towards a programmable entity. From everything we can observe in computing, this new view of a network will change the way we do networking. And SDN, for the first time, provides an industry-wide acceptance and realization.

We than argue that SDN is missing one very important aspect: the North-Bound Interface. With the NBI missing, it does not (yet) address two essential items: *providing* applications and services with the ability to use SDN to its full potential and (maybe even more important) *managing* this very new and extremely dynamic programmable network. While the original contribution of SDN is on the south-bound interface, the future success will depend on the NBI.

Exploring this future of SDN, we can foresee a very dangerous development: if all parts of a network (with the only exception being the actual physical cables and the now commoditized switches and routers) become programmable, then complete networks stacks become subject to very dynamic change. This also means that we are losing commonality, which in turn is the basis for stable network management.

In this paper, we position the RINA architecture as a way forward towards Software Defined Networking. RINA is based on proven theory, defining each layer of a network (protocol stack) using a recursive pattern, thus re-introducing commonality leading to manageability. We refer to current efforts to implement a RINA system taken by the European FP7 project IRATI.

Taking RINA as a future candidate for SDN (as in SD-Networking), we then look into three use cases and introduce the European FP7 project PRISTINE. This project investigates how a future SDN can be managed and what advantages RINA provides for this very management. The use cases are very much driven by today's demands: distributed cloud, cloud networking, and network provider requirements (or NFV). We discussed these use cases and how a RINA approach contributes to unified management of all of them.

Finally, we discuss in this paper how SDN based on RINA can be used to expose network capabilities to applications. We compare the current workflow with the workflow that will be required in the SDN future. This is a very important item to harmonize the network (now an SDN, as in a programmable entity) with the applications and services using it.

The paper also looks into what network management will look like in the future we describe. We explain the four different management points, and how they relate to a RINA-based SDN. To emphasize that this is not simply an academic exercise: the use cases and the management points this paper introduces and discusses are driven by the industrial partners of PRISTINE and their customers. While our contribution today is a discussion, our contribution in a year's time will be several prototypes solving real customer needs.

ACKNOWLEDGMENT

The authors would like to thank the Pouzin Society (PSOC) and the FP7 project IRATI for their support. The research leading to these results has received funding from the European Community's Seventh Framework Programme for research, technological development and demonstration under Grant Agreement No. 619305 for the project PRISTINE.

REFERENCES

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, and J. Rexford, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 493–512, April 2008.

[2] M. Howard, "SDN and NFV Strategies: Global Service Provider Survey," Infonetics Research, Mar 2014.

[3] (2014, May) Dynamic Service Chaining with SDN. Ericsson. [Online]. Available: http://archive.ericsson.net/service/internet/picov/get?DocNo=28701-FGC1012605/14B&Lang=EN&HighestFree=Y

[4] (2013, November) AT&T Vision Alignment Challenge Technology Survey - AT&T Domain 2.0 Vision White Paper. AT&T. [Online]. Available: https://www.att.com/Common/about_us/pdf/AT&T%20Domain%202.0%20Vision%20White%20Paper.pdf

[5] M. Weiser, "The Computer for the 21st Century," *Scientific American*, September 1991. [Online]. Available: http://wiki.daimi.au.dk/pca/_files/weiser-orig.pdf

[6] J. G. Andrews, S. Buzzi, W. Choi, S. V. Hanly, A. Lozano, A. C. K. Soong, and J. Zhang, "What will 5G be?" *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 6, pp. 1065–1082, June 2014.

[7] Networked Society. Ericsson. [Online]. Available: http://www.ericsson.com/thinkingahead/networked_society

[8] J. Strassner. (2012, September) Does sdn make my network management look fat? SDNCentral. [Online]. Available: https://www.sdncentral.com/technology/does-sdn-make-my-network-management-look-fat/2012/09/

[9] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using OpenFlow: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 493–512, 2014.

[10] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[11] Sherwood, Rob and Gibb, Glen and Yap, Kok-Kiong and Appenzeller, Guido and Casado, Martin and McKeown, Nick and Parulkar, Guru, "FlowVisor: A Network Virtualization Layer," OpenFlow Switch Consortium, Tech. Rep., October 2009. [Online]. Available: http://archive.openflow.org/downloads/technicalreports/openflow-tr-2009-1-flowvisor.pdf

[12] H. Kim and N. Feamster, "Technology improving network management with software defined networking," *IEEE Communications Magazine*, vol. 51, pp. 114–119, February 2013.

[13] N. Feamster. (2014) OpenFlow Switch Consortium. [Online]. Available: https://www.coursera.org/#course/sdn

[14] S. Albrecht, "Unifying cloud and carrier networks paves the way towards the networked society - unifying the networked society," Keynote, August 2014. [Online]. Available: http://www.fp7-unify.eu/files/fp7-unify-eu-docs/Results/presentations/ITS%202014%20Key%20Note-PA14.pdf

[15] ETSI, "Network Functions Virtualisation (NFV); Architectural Framework," ETSI, Tech. Rep. GS NFV 002, v1.1.1, October 2013. [Online]. Available: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf

[16] L. Fallon, J. Keeney, and S. van der Meer, "Towards real-time management of virtualized telecommunication networks," in *1st International Workshop on Management of SDN and NFV Systems*, 2014.

[17] J. Nolan, F. Goldstein, and J. Day, "The last waltz and moving beyond tcp/ip," *The Journal of the Institute of Telecommunicatins Professionals*, vol. 5, no. 3, 2011. [Online]. Available: http://rina.tssg.org/docs/ITP_vol5_p3_42-50.pdf

[18] J. Day, *Patterns in Network Architecture: A Return to Fundamental*. Prentice Hall, January 2008.

[19] (2014) Pouzin Society (PSOC). [Online]. Available: http://pouzinsociety.org/

[20] J. Day, I. Matta, , and K. Mattar, "Networking is ipc: a guiding principle to a better internet," in *2008 ACM CoNEXT Conference*, 2008.

[21] (2014) Investigating rina as an alternative to tcp/ip. IRATI Consortium. [Online]. Available: http://irati.eu/

[22] IRATI. (2014) Rina education - a collection of resources. [Online]. Available: http://irati.eu/education/

[23] P. Society. (2014) Rina - a collection of resources. [Online]. Available: http://rina.tssg.org/

[24] (2014) Programmability in rina for european supremacy of virtualised networks. PRISTINE Consortium. [Online]. Available: http://ict-pristine.eu/

[25] J.-P. Smets-Solanes, C. Cerin, and R. Courteaud, "Slapos: A multi-purpose distributed cloud operating system based on an erp billing model," in *IEEE International Conference Services Computing*, 2011.

[26] J. Chroboczekl, *The Babel Routing Protocol*, RFC 6126 (Independent Submission), Internet Engineering Task Force Request for Comments 6126, Apr. 2011. [Online]. Available: http://tools.ietf.org/html/rfc6126.txt

[27] ETSI. (2013, October) Network functions virtualisation. Update White Paper. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper2.pdf

[28] IETF. (2014) Service function chaining working group. Charter for Working Group.

[29] IEEE, *Standard for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks - Amendment 8: Shortest Path Bridging*, 802.1aq, Institute of Electrical and Electronics Engineers Standard 802.1aq, Apr. 2012. [Online]. Available: http://www.ieee802.org/1/pages/802.1aq.html