Software Defined Networking Demands on Software Technologies

T. Galinac Grbac*, C.M. Caba**, J. Soler **
* University of Rijeka/Faculty of Engineering, Rijeka, Croatia
** Technical University of Denmark/Networks Technology and Service Platforms, DTU Fotonik, Lyngby, Denmark
tihana.galinac@riteh.hr, {cosm, joss}fotonik.dtu.dk

Abstract - Software Defined Networking (SDN) is a networking approach based on a centralized control plane architecture with standardised interfaces between control and data planes. SDN enables fast configuration and reconfiguration of the network to enhance resource utilization and service performances. This new approach enables a more dynamic and flexible network, which may adapt to user needs and application requirements. To this end, systemized solutions must be implemented in network software, aiming to provide secure network services that meet the required service performance levels. In this paper, we review this new approach to networking from an architectural point of view, and identify and discuss some critical quality issues that require new developments in software technologies. These issues we discuss along with use case scenarios. Here in this paper we aim to identify challenges for further evolution of software technologies in addressing problems of network evolution. Our view is based on the need for strong integration of network and software technologies, and therefore we establish the main focus of the paper on discussing SDN demands on software technologies. The main contribution is in categorization of open research problems and presenting ideas and opening new opportunities for future research.

I. INTRODUCTION

Software Defined Networking (SDN) is a paradigm that is related to idea of offering the network resources to end users as a service (NaaS) over an open Application Programming Interface (API). In this paradigm, from the network operator perspective, the key differential element is to replace industrial standard Command Line Interface (CLI) for managing directly network devices with an open interface, which can be used to programmatically perform device management. Today, we have two protocols NETCONF [1], proposed by IETF, which provides mechanisms to install, manipulate, and delete the configuration of network devices, and Open Flow[2], proposed by ONF, as communications interface between the control and forwarding layers of an Software Defined Network (SDN) architecture. Introduction of open protocols enables faster evolution and interoperability among equipment of different vendors. Thus, leads to an idea of separating switching hardware and control logic as is presented in Fig. 1.

In OSI layer two and layer three implementations of the modern transmission networks there is a plethora of control protocols allowing for autonomous switching and routing of Internet traffic. Autonomous control is achieved by their distributed communication relying on continuous tracking of surrounding network topology. Thus significant processing power of network equipment is spent on spreading common topology view among these network nodes (according to some research studies more then 30%). Also, transmission overhead and link utilization may represent also significant issue. With the continuous trend of network scale, creating larger transmission chains, this current autonomous control approach becomes more and more inefficient and leading to unacceptable performances and convergence time exceeding conventional constraints. As network grows, more and more centralized approaches to network management are needed. The main problems identified are with **responsiveness**, **reliability** and **scalability**. A solution that is getting more and more attention nowadays is introduction of aforementioned layering approach with centralization of control logic. The aim of separating control logic from the switching logic into two distinct network planes is to decouple the forwarding and device operating functions from the network control logic. This new approach increases the efficiency in handling a dynamic network, and provides better adaptation to changing traffic patterns.

The idea of fully integrated network solution with separation of control logic into separate layer from pure hardware switching layer without logic, just based on dynamically changing data table, is far away from realization. Although, it looks simple it requires significant virtualization of network management activities and procedures that are not so simple in the current state of the art. Therefore, an endeavor has to be invested into simplification of existing network management operations providing backward compatible, easy to scale and performance wise solutions.

The aim of this paper is to introduce into concepts of Software Defined Networking and explore and discuss open research problems on SDN software technology. For this purposes we survey work on SDN software technologies and on concrete use cases we identify and explain main open research problems in that area. Finally, main contributions of the paper are in introducing new ideas on future research that is to be considered in developing of SDN software technologies.

Identify applicable sponsor/s here. If no sponsors, delete this text box.

II. SDN ARCHITECTURE

The basic key elements of SDN are separation of network control logic from network hardware operation and simplification of network device logic. This enables automatization of network management processes and logic above the control layer, centralization of network control logic, and openness of network resources through open standards and to end users.

Figure 1 captures the overall SDN architecture, which consists of three planes (layers): data plane, control plane and application plane. The data plane comprises the forwarding devices. At the control plane, the logic is implemented in a software platform termed SDN controller (SDNC). The SDNC's interface towards Data Plane is realized through the southbound API (*Application Platform Interface*) [3], see Fig 1. The SDNC platform contains a logical representation of the underlying network resources, and several control functions that implement operations and functions for the network. The capabilities built into the network control plane are exposed to the application plane through the Northbound API that is SDNC's interface for communication between Application and Network Control Plane. The applications, located in the application plane, leverage the northbound interface to implement higher layer service logic.

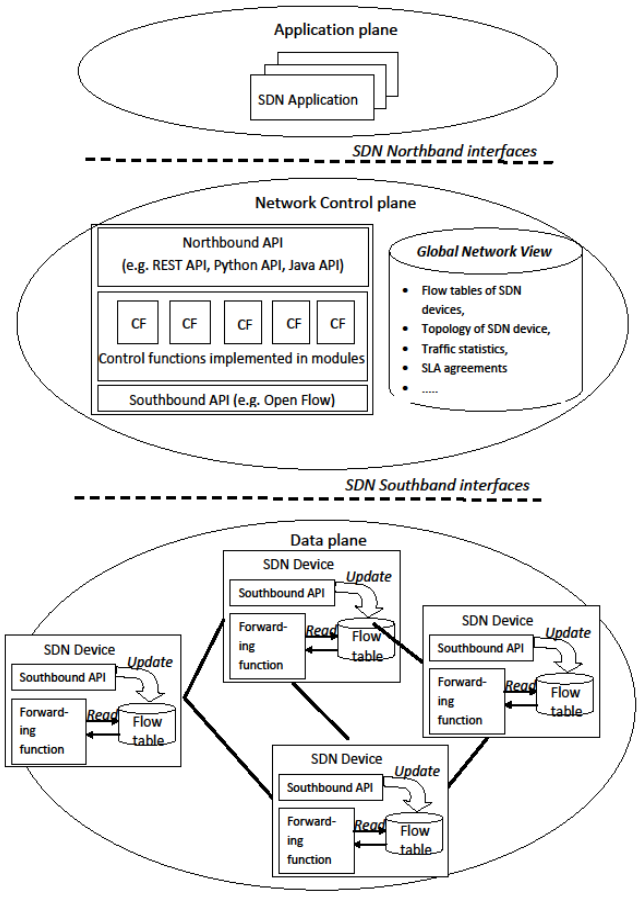

Figure 1. Overview of Software Defined Network Architecture

A. SDN Northbound Interfaces

The SDN northbound interfaces, as is defined in [3], specifies the functions exposed by the SDN control plane, for programming of SDN applications and services and it is implemented as an Application Programing Interface (API). There can be various implementations for the northbound API, depending on the network control plane and the requirements of the network services built on top of it. Figure 1 shows some possibilities for such implementations (e.g. REST, Java, etc.). These implementations fall under three main categories:

- Controller APIs that use a general purpose language (e.g. Java, C, Python).
- APIs based on web services (e.g. REST).
- APIs based on Domain Specific Languages (DSL).

The first category contains the most common and used APIs. They allow the developer to write new service modules integrated in the controller platform (Figure 1). The variety of languages depends on the implementation language of the controller platform. Some examples of controllers are: NOX in C++, POX in Python, Floodlight, ONOS and OpenDaylight in [4], [5], [6], [7], [8]. The northbound APIs in this category lead to tight coupling of the services (control functions) with the controller platform.

The second category of APIs is based on web services. Some controllers offer a REST API through which applications can get information about the state of the network, and send control messages to the switches. This type of APIs emphasizes simplicity and modularity: the network services may be programmed in any language as long as they comply with the REST interface. They are well suited for integration with external platforms, using service oriented architectures. The main disadvantage is that it may be difficult to expose the full set of capabilities of the control plane as REST APIs. While the control plane and the southbound interface are generally event-driven (network generated events), it is harder to implement an event driven REST API thus to preserve the same programming model towards the SDN applications. There are efforts in the open source community to provide event notifications through the northbound interface [6], although this is challenging. The RESTCONF protocol [9] tries to establish a standardized method for exposing the control plane resources (e.g. devices, flows, etc.) to the SDN applications using an event-driven paradigm (including event notifications and remote procedure calls).

The third category of APIs is based on domain specific languages. Their main purpose is to ease the task of expressing network policies by using high level abstractions. Examples of research work in this area are Netcore, Frenetic, and Pyretic [10], [11]. A network control plane that exposes such an API, takes as input a policy describing the expected network behavior, and compiles the policy into low level rules (e.g. Open Flow entries) that are installed in the forwarding devices. This programming model makes application development easier by moving the complexity from the applications to

the control plane policy compiler. The policy compiler ensures that the behavior enforced in the data plane is correct and does not lead to inconsistent traffic treatment (due to possible conflicting policies).

One of the challenges regarding the definition of northbound APIs is choosing the correct level of abstraction for the network resources exposed through the APIs. If the abstractions are very low level, then the application development becomes increasingly complex. On the other hand, high level abstractions make development easier but they may become less powerful (i.e. decrease visibility and control over the resources).

B. Southbound Interface

The southbound interface defines the communication between the network control plane and the data plane (Figure 1). The OpenFlow (OF) protocol has been standardized in 2009 and it is currently the most used protocol implemented on the SDN southbound interface. OF is based on abstraction of forwarding logic using flow tables. Inside a forwarding device, the forwarding function is performed based on the OF entries in one or several flow tables (Figure 1). An OF entry is a *match* and *action* tuple. The *match* part defines the headers to match against the packets arriving at the forwarding device, while the *action* part defines the actions to apply to the matching packets. In its first version, OF had limited capabilities, covering mostly Ethernet, IP, and transport headers [2]. In later versions several features have been defined, making OF more versatile:

- Pipe-line processing of packets using multiple flow tables. This mechanism enables applying a series of actions for a packet.
- Tunneling support using Multi-Protocol Label Switching (MPLS)[12] and Provider Backbone Bridge (PBB) [13].
- Support for IPv6.

Several other extensions have been defined for the OF protocol in order to support circuit switched and optical networks. While OF enables controlling the forwarding behavior in the data plane and collecting statistics about the network, it does not make it possible to configure the devices. Examples of protocols that enable configuration and management of devices are NETCONF, SNMP, and OF-Config [1] [14] [15]. These protocols enable for example configuration of operating parameters of the devices, such as queues, ports, etc.

III. USE CASE SCENARIOS

Two exemplary case scenarios are chosen here to illustrate the advantages provided by SDN in managing current network services and infrastructures.

1. Network Slicing / Virtualization

A network infrastructure under the supervision of an SDN controller (SDNC) can be "shared" by multiple users without mutual knowledge or impact, creating de-facto "virtual networks". This is a likely solution for "multitenant" operations where network infrastructure resources can be dynamically allocated and modified among the different tenants via the centralized and programmatically control of the SDNC. While this type of network "slicing" can be achieved by the physical separation and control of ports and queues in the forwarding devices, a more scalable and flexible solution can be achieved if the SDNC becomes the manager for tag-based (overlay) conventional virtual networks, i.e. by managing VLAN (Ethernet) tags or MPLS labels.

2. Data Center & Virtual Infrastructure

This is very much exploited currently in different commercial solutions targeting Datacenter (DC) networks, where computer resources virtualization has become the de-facto standard mechanism to maximize the utilization of equipment and distributing its cost. DC network topologies are based on massive tree structures of servers rooted by common switches in share physical collocated racks (i.e. TORs: Top of the Rack switches). TORS are as well interconnected in tree architecture towards gateways connecting to external networks. While traditionally "leaf" nodes in these tree topologies were mere hardware servers, in today virtualized environments they are virtual machines (VMs), instantiated when and where needed by virtualization hypervisors. These VMs, in order to enable connectivity within the virtualized network are linked via virtual switches, which can be controlled by an external SDNC. This enables to create overlay networks, as the ones mentioned previously, within the DC to represent different tenants (customers) or applications. DC network management challenges, such as dynamic traffic migration (i.e. when copying or migrating a VM from one physical server to another) or inclusion-deletion to-from different virtual (overlay) networks within the DC, become extraordinarily simplified when based on a centralized-programmatic SDNC operation, rather than a manual configuration-reconfiguration from a human (fail-prone) manager.

IV. SOFTWARE TECHNOLOGIES

The introduction of network virtualization enabled further development and flexibility of networking in cloud environments. However, SDN aims to centralize network control, thus it aims to *softwarify* network engineering theory into network services offered to its users leading to easier and more efficient network management. However, numerous problems exists and still more work has to be done to define network engineering theory that would be ready for network softwarisation.

Here we aim to introduce some of the problems and open areas for research but we are not aiming to review all the state of the art in this field.

V. OPEN PROBLEMS

Movement to the new SDN paradigm, aiming to *softwerize* the network, is full of challenges. There is a growing body of research in that direction. Some well established top journals have devoted a special issue to SDN. There have been three HotSDN conferences with

Identify applicable sponsor/s here. If no sponsors, delete this text box.

continuously increasing number of papers submitted. A new conference started on Software Defined 5G Networks. Workshops are addressing particular topics in SDN such is for example Workshop on Architectures and programming paradigms for emerging 5G Networks. Numerous papers have already been written and published and number is continuously increasing. One of the biggest concerns is how can software technologies help in this migration and where we need their help. Main challenges with network virtualization are addressed per each identified use case scenario in [16]. Here in this paper we aim to identify challenges for further evolution of software technologies in addressing problems of network evolution. We survey problems addressed by numerous authors on SDN software technologies and discuss them here in several categories. Also, important to note is that we are not aiming to review all the existing literature on challenges and provided solutions. Here we just describe open problems and occasionally provide reference for better illustration of problem.

First problem that we address here is the consistency problem between the logical representation of the network resources at the network control plane, and the physical resources in data plane. This consistency problem is associated with liveness properties. This problem is twofold. If we strive to keep consistency then we will sacrifice responsiveness and vice versa. A consistent network model at the control plane requires that well defined synchronization procedures are executed frequently in order to update the network control logical model with information from the data plane. This leads to increased resource consumption at the SDNC side and introduces delays on the control channel, thus decreases the overall responsiveness at the control plane. On the other hand, optimizations may improve responsiveness but may also introduce uncoordinated behavior that in the case of large network complexities, is leading to a non-reliable system, with routing loops and black holes. Optimizations as shortcuts are not welcome solution in such complex systems. System scaling is limited with distributed relationship between logical and physical resources and maintaining their mutual relationship is not trivial [17]. Software technology research aims to identify set of representative relations, that are easy to manage and maintain, investigate specific trade-offs that are dependent on controller implementations and to find alternative mechanisms on delays or reliability. Consistency effect is evaluated from different perspectives and vast amount of software technologies has been proposed to address this problem and still this is a hot research area widely involving computer science and complex network theory.

To achieve high scalability, flexibility and availability it is identified as mandatory to implement redundancy at control plane. In [18] experimenting with different topologies it is identified that one controller location is often sufficient to responsiveness requirements but not for fault tolerance requirements. Some researchers has come with statement that the only possible way to secure highly available but reliable platform is to introduce redundancy and distribution of control logic. However, in that scenario we introduce completely new branch of problems dealing with synchronization and concurrency issues while handling concurrent policy updates [19]. Here we face with full complexity of programming network control function with high reliability and availability demands while capable for executing numerous concurrent processes for variety of different users. Majority of knowledge in this domain has already been collected within industry such is for example Ericsson knowledge base developed in more than 40 years of evolution of AXE based telephone exchange that in its *AXE 810* version that implements control logic as standalone node that is central in next generation networks for handling control (signaling) of end user traffic in resource network layer. This switch is central to many core network implementations of Next Generation Networks (see 3GPP standards) and is reliable, robust and easy to scale solution. However, most of the solutions are not implemented to be used at runtime as is required in new network evolution phase (5G networks). The synchronization and concurrency issues are big concern of these networks. As communication is a central element of these networks and aiming to *softwarify* network engineering theory we may need to control harmonies execution of numerous industries specific hardware related policies and numerous interacting industry standards. This is actually one of the biggest concerns of further network evolution. For that reason we may need new abstractions for structured communication - centered programming and to specify and implement program communication safe software. One promising example is behavioural type theory that introduces new behaviuoral types for describing interactions involving multiple peers and that abstracts these interactions as a global scenario [20], [21]. Message exchange among concurrently running multiple peers are verified at global level thus enabling independent certification of each party's local implementation if it is in correspondence to global behaviour. Based on that theory a lot of interesting approaches has been presented at workshop on *Programming Language Approaches to Communication and Concurrency cEntric Systems (PLACES)*. One good example is Scribble programming language [22] that allows certification of global protocol interaction and projection onto local protocol implementation. Except the tools for editing, verifying and projecting, there is work in progress of development of numerous libraries that allow its integration with some general purpose languages such are java or python.

In centrally controlled network architecture that is continuously growing, its management becomes extremely complex and security vulnerabilities becomes more complex and error prone. Data integrity and confidentiality for information exchange in a network with dynamic policies require innovation in the area of cryptographic algorithms or authentication certificates. Some cyber security controls, that would be implemented as part of software tehnology for next generation systems

such as access control, network isolation or monitoring are areas under investigation.

Another problem is with central management of global network view. This task requires huge amount of network data to be effectively stored into database, restored and up to date. Problem of storing of huge amount of data as well as effective information capturing from this big data volumes is a huge issue. One approach is usage of hierarchy abstraction. In [23] proposal is to use hierarchical policies that show positive performance results.

Expression of configuration protocols such as Open Flow and Netconf is limited. A numerous studies question their flexibility to different configuration needs, and their expression abilities in relation to the performance requirements [24]. Here we have tight interaction between telecommunication and computer science profession in defining right abstractions that would provide expressive protocol and its implementation language for efficient, reliable but secure communication between abstract network notation and physical resources. There is continuous work in progress in standardization of these protocols and its implementations as well as identification of new avenues for further evolution in configuration protocols and languages.

Modeling abstractions, software organization at control but also at device level may have significant influence of service performance offered by SDN network. This is especially interesting to observe in relation to system potential to scale. For example, an earlier study show that splitting of architecture in Mobile Switching Centre has significantly degrade service performances and scaling abilities, [25]. Software modelling methods have to be revised with software dynamics in mind. A software engineering modeling tool that aims to model how software application with intensive dynamic scaling requirements has to be distributed in respect optimized resourse costs has been proposed by [26]. An object oriented programming approach implementing type system aiming to optimize resource usage costs by governing software distribution within multicore systems has been proposed in [27].

Architectures that rely on open standards and open source code community benefit in terms of progressive development and prototyping. Moreover, an integrated view that is maintained among various industries through open community turns out with positive interoperability characteristics. However, there are numerous drawbacks. Open community is good choice for simple prototyping solutions without strict and formal rules, based on easy and general purpose languages. But, in case of complex system there are numerous studies that proved that human factor should not be neglected. For example, programming language *Erlang* contains some interesting concepts that are developed assuming unreliable software. The right balance between advanced software technologies that human should consume while developing software and software technologies that accepts not perfect world has to be achieved. One way to deal with this issue is to develop certification technology for software packages that are to be deployed into complex network that would be based on using built in typecheckers.

The way how system evolves very much influence its further ability to scale, further develop, maintain required reliability levels, etc. Learing from existing complex systems that evolved over decades may bring some fruitful knowledge. Identification of software behaviour and identification of autonomous modeling techniques that could be easily implemented into network or development environemnt may be used to introduce discipline in SDN network in open source environment.

VI. CONLUSION

Software Defined Networks have been developed and invented as new paradigm in telecommunication world but it is highly based on computer science theories. Numerous efforts that were invested within computer science field developing theories in formal languages and verification tools have finally show their full practical benefit. Developing complex software systems such a modern telecommunication networks become extremely challenging task. Add hoc shortcutting engineering solutions is not desirable any more but we should strive on systematized solutions. The best known systematization technology is governed by sound formalisms that are implemented within technology that humans use to engineer complex systems. In the SDN world the need for systematization of existing network and traffic engineering knowledge is evident. And this is still not enough; we need this systemized knowledge packed in nice formal packages that will be simple and easy to integrate in complex system that we develop and use. For this future step there is need for diverse skills and competences, a huge experience and practice base of engineering existing networks, theoretical sense for knowledge systematization and representation in meaningful way and again engineering knowledge to apply systematized solutions in practice. Moreover, expertise from different knowledge domains are identified as needed. Just to mention a few, network and traffic engineers, reliability engineers, computer scientist, mathematicians, and complex network theoreticians. Software technologies just have to be developed for new networking paradigm used for developing complex software systems and lot of efforts has to be invested prior final solution. Here in this paper we survey key SDN problems, demonstrate details on real use case examples and based on that we identify main categories of existing research in SDN software technology. Furthermore, we open some new avenues by introducing some ideas for future research in SDN software technology.

ACKNOWLEDGMENT

The work presented in this paper is supported by COST

action 1304 Autonomous Control for a Reliable Internet of Services (ACROSS), COST action 1201 Behavioural Types for Reliable Large Scale Systems (BETTY), the research grant 13.09.2.2.16. from the University of Rijeka, Croatia, and the research project project "SDN: applicability and service possibilities" at DTU.

REFERENCES

[1] IETF NETCONF Network Configuration protocol, RFC 6241, http://tools.ietf.org/html/rfc6241, retrieved on January 2015.

[2] ONF, Open Flow Switch Specification v 1.05. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf retrieved on January 2015.

[3] ONF SDN Architecture, Issue 1 (June 2014). https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/TR_SDN_ARCH_1.0_06062014.pdf, retrieved on January 2015.

[4] http://www.noxrepo.org/ , accessed on January 2015.

[5] http://www.projectfloodlight.org/floodlight/ , accessed January 2015.

[6] http://www.opendaylight.org/ , accessed January 2015

[7] P. Berde, et al., "ONOS: Towards and Open, Distributed SDN OS", Proc. HotSDN , 2014.

[8] http://onosproject.org/, accesed on January 2015

[9] Bierman et alt., ETF RESTCONF Protocol, http://www.ietf.org/archive/id/draft-bierman-netconf-restconf-04.txt, retrieved on January 2015.

[10] http://www.frenetic-lang.org/ , accessed January 2015,

[11] C. Monsanto, N. Foster, R. Harrison, D. Walker, "A compiler and run-time system for network programming languages", ACM SIGPLAN Notices - POPL '12 , January 2012.

[12] Rosen et alt. IETF RFC 3031- Multiprotocol Label Switching Architecture. http://tools.ietf.org/html/rfc3031 , retrieved on January 2015.

[13] IEEE802.1Qay- Provider Backbone Bridge Traffic Engineering http://www.ieee802.org/1/pages/802.1ay.html , accessed on January 2015

[14] Harrington et. al. IETF RFC 3411 – An Architecture for Describing Sinple Network Management Protocol Management Frameworks http://tools.ietf.org/html/rfc3411, retrieved on January 2015.

[15] ONF OF-Config v1.2. OpenFlow Management and Configuration Protocol.https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf , retrieved on January 2015.

[16] Network Functions Virtualization (NFV) Use Cases, ETSI GS NFV 001 v1.1.1 (2013-10). http://www.etsi.org/deliver/etsi_gs/NFV/001_099/001/01.01.01_60/gs_NFV001v010101p.pdf , retrieved on January 2015.

[17] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann. "Logically centralized?: state distribution trade-offs in software defined networks," In *Proc. of the first workshop on Hot topics in software defined networks* (HotSDN '12), ACM, 2012, pp. 1-6, 2012.

[18] B. Heller, R. Sherwood, and N. McKeown. "The controller placement problem", In *Proceedings of the first workshop on Hot topics in software defined networks* (HotSDN '12). ACM, New York, NY, USA, pp. 7-12, 2012.

[19] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "Software transactional networking: concurrent and consistent policy composition," In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (HotSDN '13), ACM, New York, NY, USA, pp. 1-6, 2013.

[20] Kohei Honda, Nobuko Yoshida, and Marco Carbone, "Multiparty asynchronous session types", In *Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (POPL '08). ACM, New York, NY, USA, pp. 273-284, 2008.

[21] Lorenzo Bettini, et al. Global Progress in Dynamically Interleaved Multiparty Sessions. In Proc. of *Conference on Concurrency Theory* (CONCUR 2008), volume 5201 of LNCS, pp. 418–433. Springer, 2008.

[22] Scribble programing languague, http://www.scribble.org/, Accessed Feb 2015.

[23] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Hierarchical policies for software defined networks. In *Proc. of the first workshop on Hot topics in software defined networks* (HotSDN '12). ACM, New York, USA, 37-42, 2012.

[24] A.Voellmy, H. Kim, and N. Feamster. "Procera: a language for high-level reactive network control," In *Proceedings of the first workshop on Hot topics in software defined networks* (HotSDN '12). ACM, New York, NY, USA, pp. 43-48, 2012.

[25] Galinac, Tihana, Huljenić, Darko; Influence on basic call set-up by integration of packet and circuit switched Core Network *In Proceedings of the International Conference on Computers in Telecommunications of the 25th International Convention* (MIPRO 2002) Opatija, Croatia, 2002.

[26] Tanković, Nikola, Galinac Grbac, Tihana; Truong, Hong-Linh; Dustdar, Schahram, Transforming vertical web applications into elastic cloud applications *In Proc. of International Conference on Cloud Engineering* 2015, Phoneix, USA. 2015.

[27] Franco, Juliana, Drossopoulou Sophia; Behavioural types for non-uniform memory accesses, *In Proc. Of Programming Language Approaches to Communication and Concurrency cEntric System, 2015, London, UK, 2015.*