2015 IEEE 29th International Conference on Advanced Information Networking and Applications

Introduction and Analysis of SDN and NFV Security Architecture (SN-SECA)

Danilo V. Bernardo
Sydney, AUSTRALIA
e-mail:bernardan@gmail.com

Bee Bee Chua
University of Technology Sydney
Sydney, AUSTRALIA

***Abstract*—There have been a few literature published about the security risks expected on the implementations of SDN and NFV (SN), however, no formal Security Architecture with practical attributes was proposed until recently. The first of its kind SN-SECurity Architecture (SN-SECA) was presented as an IETF draft. This draft presents the architecture with specific ascription to ensure effective security evaluation and integration on the SDN/NVF designs and implementations. This paper briefly introduces the proposed architecture and employs methods to analyze and verify its underlying security attributes. A unified method to review SN-SECA through symbolic analysis previews traffic process flow behavior across an infrastructure with SDN and NFV frameworks. The result of this work highlights the fundamental but important role of each attribute and its flow, and overall viability of the proposed architecture for SDN and NFV that protractedly useful to security practitioners.**

Keywords—SDN; NFV;SN-SECA; OpenFlow; Security Architecture; rewrite; symbolic analysis

I. INTRODUCTION

Software Defined Network (SDN) and Network Function Virtualization (NFV) frameworks [3] have been creating paradigm shifts across industries. In recent years, these two novel frameworks attracted interests throughout many academic institutions and communities of practice.

They may be considered disruptive to major network infrastructures operating on status quo, but SDN and NFV offer network resiliency, manageability, and, most importantly, affordances –lowering long-term operational expenditures when implemented properly. They create opportunities for innovation that result from providing platforms where key players from networks, security, and software groups develop new controllers, APIs, networks, and technologies. However, new innovations come with associated risks and security issues.

A. SDN

In 2012, it was widely speculated that Google implemented SDN and OpenFlow into their networks, and created their own OpenFlow-enabled switches due to the limited vendors supporting this protocol. This speculation and clear benefits of deploying SDN and OpenFlow have since gained significant interests across many industries.

To understand SDN is to review the OSI layer, where concepts of abstraction and separation, are similar to tiering and layering, which determined by the layers of the stack.

In SDN, control and data planes are separated, centralizing the control and programmability of the network. To connect applications across the upper and lower planes, Application Program Interfaces (APIs), which a few were successfully standardized (eg. OpenFlow) are utilized.

The components of the SDN framework are composed of Northbound and Southbound APIs.

Northbound APIs achieve abstraction to the top of the framework while the Southbound APIs achieve the same at the bottom of the framework.

Additionally, East and Westbound APIs, which are formally introduced in the IETF draft [3], are used for horizontal communications across devices, systems, or software on a given plane.

Summarily, the Northbound and Southbound APIs are used for vertical connections, while the East and Westbound APIs are for horizontal connections. Notably, the application tier /plane can host a variety of business applications and application-based security systems.

The Application plane normally composes of application driven engines. The control plane on the other hand, composes controllers or orchestrators, where routing and security policies are implemented. The data plane is basically the infrastructures composing network devices, such as routers, switches, and security devices. Each plane can be virtualized as and when needed.

Initially, the implementations of SDN targeted campuses, Data Centres and Cloud, but eventually expanded to other areas and adapted by service and network providers.

B. NFV

NFV was formalized in 2012. Its goal is to relocate network functions from dedicated hardware appliances to generic servers. It is initially intended for routers, firewalls, and gateways, but can be expanded to include load balancers and other intermediary devices [3].

1550-445X/15 $31.00 © 2015 IEEE
DOI 10.1109/AINA.2015.270

IEEE computer society

Unlike SDN, NFV does not have a specific protocol. However, both SDN and NFV create open innovation by third parties, reducing capital and operational expenditures.

C. Security Architecture

A formal security architecture (SN-SECA) that overarches important aspects of SDN and NFV's frameworks was proposed to aid practitioners integrate security in their SDN/NVF designs and implementations. This architecture, which is considered first to be proposed through IETF draft [3], can be elaborated and modified as the frameworks mature over time.

SN-SECA was developed based on general practices (use-cases [22,24]) on traditional networks across different industries. It is presented with supplemental information on the attributes, which can provide a basis for basic, if not comprehensive security profile for each plane within SDN and NFV. Essentially, the idea is to use a common baseline design that, on one hand, provides a sufficient level of protection of systems and devices within the plane, but which, on the other hand, is deemed practical and deployable to infrastructures that operate on SDN and NFV.

D. Contributions

The following contributions to the analysis of the proposed SN-SEC are presented.

- The formalization of SN-SECA for inductive analysis and automata of its security attributes by proving connections between selected trace properties of implementation and non-selected theoretic properties standard in the literature;
- Introduction of a type of retrofit: a revision and replication. This type of retrofit implies changes to planes of the architecture which must be articulated for a validity requirement to be constructed and evaluated. As is true in architecture design, the paper highlights a method, which traces the (*TP*) flow that does not limit validation scenarios surrounding the intended effect of the architecture to specific implementations.
- Security validity to support SDN /NFV implementations.

II. Organization

Firstly, the paper presents the architecture that published recently as an IETF informational draft [3]. Secondly, it describes selected methods of analyses, and how these are employed to review the architecture design. Then lastly, it describes and illustrates the layers of validating the architecture and its design; and discusses the concept of architectural retrofit, which is the process of altering the architecture after it has been put into operational use.

III. Architecture

The development of the architecture was based on the use-cases of traditional security implementations across many industries. We use use-cases [22,24] on SDN/NFV, which are limited at this stage.

The industries discussed in this work, are not limited to academe, financial, government, manufacturing, retail and telecommunications, where specific infrastructures and business requirements certainly vary depending on their existing networks and on their maturity to achieve a consolidated security overview of infrastructures to form an architecture that is generalizable to most community of practices.

Attributes of the proposed architecture are composed of APPS – Applications, Northbound APIs, Application Tier, Extensive Validation, East and Westbound APIs, Control Plane, Data Plane, SDN Controller, Open Flow enabled devices, Governance and Frameworks, other virtual and physical devices, and Best practices.

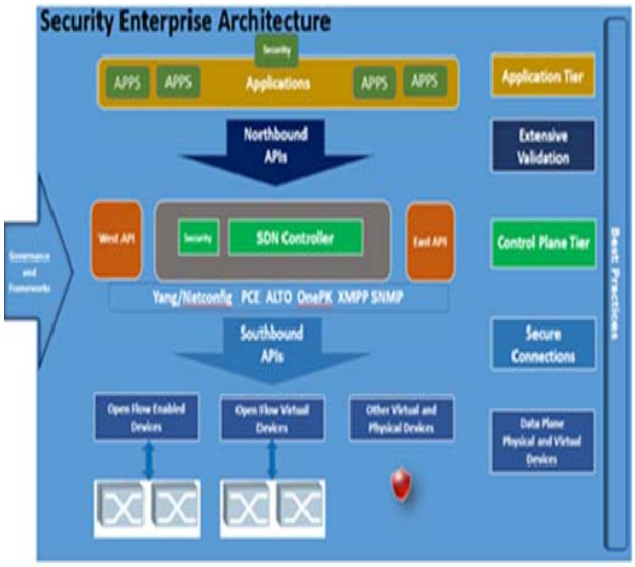

Figure 1. Plane-to-Plane High-level SN-SEC Architecture. In this architecture, the application plane hosts security functionalities to protect applications. The security schemes that can be implemented on this layer are application-based security such as application firewalls and IPS.

The Northbound APIs can be implemented for applications to work with SDN controller, thus in absence of existing standards, an external validation is necessary. On the SDN control plane, various attributes can be introduced, provided they enforce security policies through OpenFlow to the virtual devices. SDN controllers utilized as orchestrators and unified management systems can enforce security to other devices through other Southbound APIs.

The link between the controller and the remaining plane can be protected using IPSec (securing end-to-end), TLS or HT*TP*S. Other intermediary security devices, such as IPS/IDS, can be deployed across the planes.

The following are the risks identified on SDN/NFV based on the traditional network designs.

a. Adversarial traffic flows - traffic passing through network devices, interfaces, and hosts.

b. Attacks on vulnerabilities in network devices - not updated patch and version

c. Attacks on vulnerabilities in orchestrators and administrative servers - unsecure servers, lacks security on admin profile

d. Attacks on control plane communications- single point of failures, unreliable controllers and unsecure connections

The following are identified and considered new (non-traditional) risks associated with SDN/NFV implementations, and as follows:

e. Attacks on SDN controllers - unreliable software APP/SDN controller

f. Attacks on Southbound interfaces - exposed interfaces

g. Unsecure OpenFlow traffic - unencrypted /unsecure traffic

h. Unreliable North/East-West APIs - unreliable software installed across the same broadcast domain

i. Vulnerable Programming models – inadequate security involvement in the Software Development Lifecycle.

A few mitigation guidelines have been identified that must be considered when implementing SDN/NFV.

j. Hypervisors must be secure

k. Controllers must be secure

l. Hardening OS security

m. Extensive API validations

n. Extensive APPs penetration tests

o. Monitor traffic

p. Protocol must be secure

q. Session establishment protocol for communication and traffic flow must be secure

r. Multiple authentication

s. Limited time options for messaging

t. Network devices must be secure

u. Separation/segmentation of networks and subnets

These attributes (see figure 2) are introduced in the SN-SECA. They are inclusive of the security mitigations that must be considered in specific network implementations.

IV. APPROACH

In the succeeding sections, the approach and methods of analyses presented were tested and employed in other security architecture [4]. Thus follow the same logic with different variables for SN-SECA.

A. Traffic and Process (TP) Flow

In this approach, *TP* flow is represented as algebraic term. Its symbolic representation is based on the following signatures:

Left, right: *attribute* → *attribute*
Right, left: *attribute* ← *attribute*
Up, down : *attribute* ↓ *attribute*
Down, up : *attribute* ↓ *attribute*
: → *attribute*
From : *attribute x attribute* → *TP-source*
Dest : *attribute x attribute* → *TP-destination*
TP flow :*flow-source x flow-destination* → *TP flow a,b* : *attribute* →*attribute*

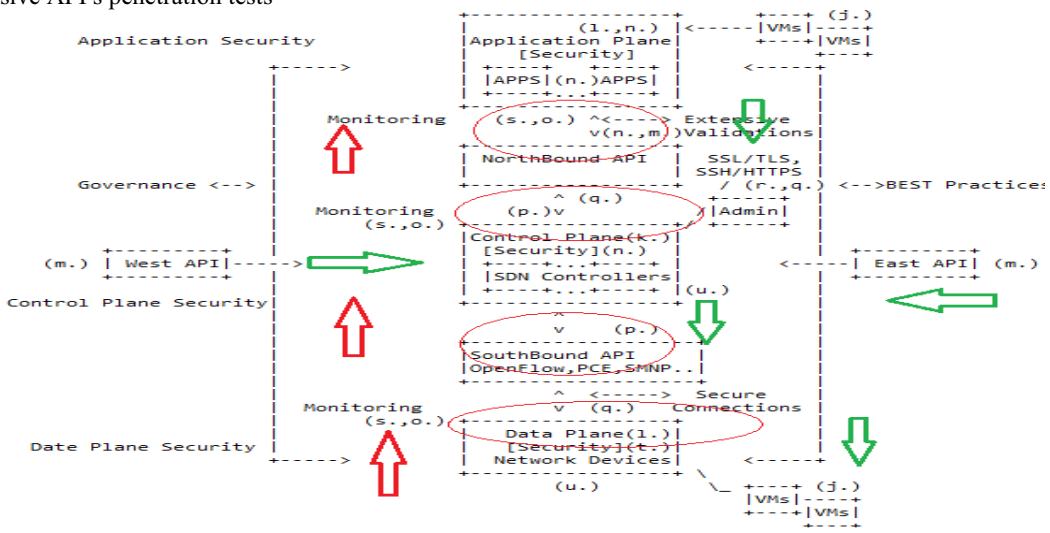

Figure 2. Practical SN-SECArchitecture [3]

Attributes: *ab(1) j, ab(2)k, ab(3)l, ab(4)m, ab(5)n, ab(6)o, ab(7)p,ab(8)q,ab(9)r,ab(10)s,ab(11)r,ab(12)t*

There are various possibilities to describe how the *TP* flow passes through the attributes. The flow, however, only relies on the given flow source. To distinguish which attributes is used, they are presented as words over {*a,b*} and a constant # (1,2..n). We limit our representation of data as *TP* flow, flow-source, flow-destination, and # to specific attributes we proposed.

Example 1.

For example the term t=ab(#). # identifies as an attribute, example ab(1) is assigned to a specific attribute j. This representation allows us to build a tree automata recognizing *TP* flow and attributes to analyze the architecture as a state.

Data are terms of sort *TP* flow for example, the term *TP* flow *(from(ab,x),dest(ab,x))* represents a *TP* flow *(from(ab(1),x),dest(ab(1),x'))* to the set of data whose flow-source comes from a security attribute ab# and flow-destination exits to the same attribute is a given flow.

In the preceding section , we define how the flow can be analyzed across ab(#) attributes (using the symbols flow-source and flow-destination) thus allows us to encode and assume a symbolic attack by appropriate tree automata[15], which we use for the analysis.

B. Architecture Traffic Flow

Furthermore, to represent the *TP* flow in the proposed architecture, we add the following symbols.

Enter or accept, Exit or reject : → *Decision*

From a rewriting point of view, the flow rewrites a *TP* flow into enter, exit or reject. However, the security attributes *(j…u)* do not necessarily reject each other, if and when one of them do not exist (i.e.,caused by its absence or by an anomaly).

Definition 1 (Attribute). An attribute is composed of three ordered rewrite systems *Prem, Processm,* and *Exitm* such that:

- flows of *Prem* are of the form of $p \rightarrow d$ where p is a linear term of sort *TP* flow and d *a* (ground) term of sort Decision;
- flows of *Prem and Exitm* are, respectively, of the form:

  From(attribute, x) → *From(attribute', x')*
  Dest(attribute, x) → *Dest(attribute', x')*

  Where attribute, x are linear terms and attribute', x' are ground terms.

Example 2. The attribute described in Example 1 can be specified as follows:

{ (From (*ab*[x],attribute))
TP flow Dest (*ab*[y],z) → accept
TP flow *(x,y)* → *exit*

- X, Y (number=1...n) and Z =attribute

Definition 2 (Semantics). For any attribute [1…n], its semantics is denoted by [*m*] and defined as follows:

[m] = [m]accept U [m]exit OR [m] reject

With R:{x→x} is the rewrite system R in which the flow x→x has been added as the last flow.

[m] accept = $\{(t,u) \in T^{TP\ flow} \times T^{TP\ flow} \mid \exists v \in T^{TP\ flow}, t \rightarrow_{Prem;\{x \rightarrow x\}} v \rightarrow_{Processm} \mathbf{enter} \wedge v \rightarrow_{Exitm;\{x \rightarrow x\}} u\}$

[m] exit $=\{(t, \mathbf{exit}) \in T^{TP\ flow} \times T^{Decision} \mid \exists v \in T^{TP\ flow}, t \rightarrow_{Prem;\{x \rightarrow x\}} u \rightarrow_{Processm} \mathbf{exit}\}$

From an abstract point of view, a attribute can be seen as a partial or total function which takes an input (data / *TP* flow) and returns either exit/reject.

V. Analysis of the Architecture

We show that this rewrite specification allows not only to automatically check properties concerning the semantics of a attribute as part of the overall architecture but also to perform structural and query analysis [1,15-20] on the architecture itself.

A.Semantics Analysis

A attribute can be seen as a decision process which associates to *TP* flow with data that could be either accept or reject/exit, and therefore the following properties need verification: *consistency,* which indicates that at most, one decision is taken for a given *TP* flow*; termination,* [1-7-10-12,14,17,19-20] which ensures that an attribute computes a decision in a finite time; and *completeness,*[1,16-20] which signifies that for any *TP* flow, the attribute returns a decision.

As discussed, by construction, any attribute that denotes a terminating and consistent decision process is a function. The completeness can be therefore defined as follows:

Definition 3 (Completeness). We state that an attribute (*m*) as part of the whole architecture (a), is a complete *iff [m],* which is a total function.

The particular shape of the flows defining a attribute allows us to represent the semantics of an attribute as a regular relation and to verify its completeness.

Proposition 1. *Completeness is decidable* [1-2,10-18,20].

Proof. The proof relies on the regularity of the relations involved in the definition of the semantics of an attribute as

part of the architecture. Since the left-hand sides of all the rewrite flows composing a particular attribute are linear and share no variable with their right-hand sides [1,16-17,19-20], we can show that $\rightarrow_{Prem;\{}$ and $\rightarrow_{Exitm;\{}$ are regular trees relations. Since the identity is a regular relation, it follows that $_{Prem;\{ x \rightarrow x \}}$ and $_{Exitm;\{ x \rightarrow x \}}$ are also regular. By composition and restriction, we obtain that $[m]^{accept}$ and $[m]^{exit/reject}$ are regular tree (functional) relations. Subsequently, [*m*] is a regular tree (functional) relation. The completeness can be tested by checking that the first projection of [*m*] covers the (regular) set of all possible incoming data transported by the *TP* flow.

In case of a complete architecture, yet selecting only the applicable attributes in either in isolation or in compositional groups, where these are used at the same time, but independently operate on a given speed, it is important to determine if the chosen attribute is applicably stronger than the others. This can be achieved, through verification, based on specifications of the attributes.

Definition 4 (Order). We define a partial order over complete attribute within a complete architecture as follows: for any m and m',$m \preceq m'$ (m' is more permissive than m) *iff* $[m]^{accept} \subseteq [m']^{accept}$. We write $m \approx m'$ *iff* $m \preceq m'$ and $m' \preceq m$. A attribute m' is thus permissive than a attribute m if it accepts all *TP* flow that m accepts. Note that $m \preceq m'$ *iff* [m]=[m'].

For the same reasons, we can decide whether a attribute is more or less permissive than the other within the architecture.

Proposition 2. *The order relation* $\preceq$ *is decidable* [1,3-5,16,18-20].

Proof. As we have already shown, for any attribute [m], $[m]^{accept/enter}$ and $[m]^{exit/reject}$ and [m] are regular relation. Consequently, the inclusion $[m]^{accept/enter} \subseteq [m']^{accept/enter}$ is decidable. Note that two attributes may have the same semantics even if their flows are d*iff*erent. This is particularly interesting since it allows to simplify and optimize the flows of an attribute [4,19-20] and check if the succeeding attribute has the same semantics as the preceding one.

B. Structural Analysis

Structural analysis [15-20] refers to the detection of so-called anomalies in (the implementations of) attribute. We look at these anomalies as properties expressed as relationships between the flows of attributes within the proposed architecture. Examples of anomalies are superseding (a flow leads to decisions contradictory to decisions of prior flows [18-20] of prior attributes), redundancy (a flow can be removed without any impact to other attributes and *TP* flow), generalization (a flow matches a superset of the set of data matched by a prior flow with a d*iff*erent decision). We should mention that although several approaches have been developed for the detection of the above anomalies, they are often intentionally introduced in order to obtain more compact or more effective flow sets. Detecting anomalies is still interesting since it can outline some mitigation. We only discuss here our approach for detecting superseding; the other anomalies can be treated in a similar way. Let us recall the definition of the superseding anomaly [18-20] in this context: we say that a attribute superseding *iff* it contains at least one flow such that all *TP* flow it allows and accepts are dropped by a prior flow. In such a case, the concerned flow is said to be superseded.

The detection of the superseded flows, as well as of the other anomalies, is based on the regularity of the sets of terms associated to a given flow. More precisely, each flow r is associated to several sets: *rec(r),* denoting the set of data matching *r; rec(r=Process$_m$)*, denoting the set of data matching r that match no S prior flow of $Process_m$ (i.e. *rec(r)*\ $\bigcup_{r'<r}$ *rec(r'))* and *rec(r=Process$_m$ [d])* denoting the set of data matching r that match no other flow of *Process$_m$* associated to the decision d. Since the left-hand sides of the processing flows are linear terms, all the sets *rec(r)* are regular; the other sets are also regular since they can be built starting from *rec(r)* and using operations which preserve regularity. Anomalies can then be detected using inclusion or emptiness tests. For example, to detect if a flow *r* is superseded, it suffices to check the emptiness of *rec(r=Process$_m$[accept/enter])* if the right-hand side of *r* is drop [18-20] and the emptiness of *rec(r= Process$_m$[exit/reject])* otherwise.

C. Query Analysis

Another kind of analysis we attempted is query analysis. This kind of analysis provides a way to assist researchers in understanding the behavior of an attribute within our architecture through deriving outputs based on pre-determined and pre-defined queries, such as "which attribute will receive the *TP* flowing from left to the right side of the architecture?" [3,7,6-9,11-16,19,20.] We introduced earlier the semantics of the attributes and the architecture as regular relation. The following therefore can be built on for any query expressed as first order formula [17-20]:

- variables, ground terms or terms whose head is symbol data or *TP* flow and whose sub terms are variables of ground terms [1-7,11-20].
- membership constraints *w.r.t.* to one of the relations defined in Definition 2 and [11-18]
- membership constraints [11-18,20] *w.r.t,* to a linear term.

All can be written into tree automaton [1-5,20] recognizing the set of solutions of the query, that is values of free variables verifying the formula as true [1-6, 11-19].

D. Retrofit

Retrofit allows continuous reviews and adjustments on the architecture after its implementations.

In this case, the complexity of the needed operations strongly depends on the representation of data and, in particular, on the representation of *TP* flow. The choice of *TP* flow as words over {*a,b*} (or equivalently as terms built from the monadic symbols a and b, and the constant #) is indeed made in order to obtain efficient implementations of the corresponding automata operations.

To simplify, consider the word automata: the correspondence with tree automata is straightforward. Due to the representation of *ab*(i…n) ranges, we are confronted with n-prefix (or simply prefix) languages, i.e. regular languages of the form or (*a,b*}*. A good property of the manipulated ranges is that corresponding minimal and deterministic automata have no loop except at their unique final state, which loops over itself for any word.

Moreover, the sets of *TP* flow of a given attribute are 1-prefix. It follows that rec(r), rec(r=Processm),…, are prefix languages. Consequently, anomalies can be efficiently detected using this approach.

VI. Conclusion

In verifying our proposed architecture, we use symbolic analysis of each attribute across the plane of *TP* flow.

We described the architecture using structural, semantics, and query analyses aside from using rewrite systems, which interpreted relevant properties through classical, theoretical and practical methods. We have shown that these approaches and analyses underscored any foreseen anomalies such as absence of the recommended attributes, or lack of secure traffic flow that require adequate processes, which can be utilized to assist in the implementation of selected attributes across the planes in the architecture.

Future work to survey better ways to understand and address security threats and vulnerabilities essential to enhance the proposed SDN/NFV security architecture is being explored.

References

[1] T. Aoto, J. Yoshida, and Y. Toyama, Proving confluence of term rewriting systems automatically. In Rewriting Techniques and Applications, pages 93-102. Springer, 2009

[2] F. Baader, F., and T., Nipkow, Term rewriting and all that. C.U.Press, 1998. [5] Balland, E., Brauner, P., Kopetz, R., Moreau, P.-E.

[3] D.V. Bernardo, "Software-Defined Networking and Network Functions Virtualization Security Architecture" Working paper IETF, 2014.

[4] D.V. Bernardo 'Network Security Mechanisms and Implementations for the Next Generation Reliable Fast Data Transfer Protocol - UDT ', PhD Thesis, 2012

[5] D.V. Bernardo. and D.B. Hoang, International Journal of Security and its Applications, "A Pragmatic Approach, Achieving Acceptable Security Attributes for High Speed Data Transfer Protocol – UDT", SERSC, Vol. 4, no. 3, 2010 ISSN 1738- 9976

[6] D.V. Bernardo and D.B. Hoang ,. Empirical Survey, Protecting Data Transfer in a High Speed Protocol for GRID, Presented at 25th IEEE AINA Conference, Singapore.

[7] D.V. Bernardo, and D.B. Hoang, "Security Requirements for UDT", IETF Internet-Draft – working paper, September 2009

[8] D.V. Bernardo, and D.B. Hoang, Formalization and Information-Theoretic Soundness in the Development of Security Architecture for Next Generation Network Protocol - UDT. 183-194, SecTech 2011, Jeju, South Korea Dec. 8-10, LNCS, Springer

[9] P. Borovansky, C. Kirchner, H. Kirchner, P.E Moreau and C. Ringeissen, An overview of elan. Electronic Notes in Th. Comp. Sci., 15:329-344, 1998.

[10] A.C. Caron, "Linear bounded automata and rewrite systems: Influence of initial configurations on decision properties" TAPSOFT '91 LNCS, Vol 493/1991, 74-89, DOI 10.1007/3-540-53982-4_5

[11] H. Comon, M.Dauchet , R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi, Tree automata techniques and applications. Available on: ht*TP*://www.grappa.univ-lille3.fr/tata, 2008.

[12] E. Contejean, A. Paskevich, X. Urbain, P. Courtieu , O. Pons and J. Forest, PAT, an approach for certified automated termination proofs. In ACM SIGPLAN Work. on Partial evaluation and program manipulation, pages 63-72. ACM, 2010.

[13] I. Durand, Autowrite: A tool for term rewrite systems and tree automata. Electronic Notes in Th. Comp. Sci., 124(2):29-49, 2005.

[14] I. Durand, and Meseguer,J ., A Church-Rosser checker tool for conditional order-sorted equational Maude specifications. pages 69-85. Springer, 2010.

[15] G. Feuillade, T. Genet, and T. Viet Triem, Reachability analysis over term rewriting systems. 33(3):341-383, 2004.

[16] J. Giesl, P. Schneider-Kamp, and R. Thiemann, AProVE 1.2: Automatic termination proofs in the dependency pair framework. In Intl Joint Conf. on Automated Reasoning, pages 281-286. Springer, 2006.

[17] O. Fiedrich, "On the connections between reqriting and formal language theory " REWRITING TECHNIQUES AND APPS,11th International Conf RTA 2000, Nowrich UK, July 20000 LNCS 1631/1999, 672,

[18] F. Jacquemard, Decidable approximations of term rewriting systems. In Rewriting Techniques and Applications, pages 362-376. Springer, 1996.

[19] J.P. Jouannaud and C. Kirchner, Solving equations in abstract algebras: a flow-based survey of unification. In Computational Logic: Essays in Honor of Alan Robinson, chapter 8, pages 257-321. The MIT-Press, 1991.

[20] C. Kirchner, H. Kirchner, and A. de Oliveira, Analysis of rewrite-based access control policies. Electronic Notes in Th. Comp. Sci., 234:55-75, 2009.

[21] M. Korp, C. Sternagel, H. Zankl, and A. Middeldorp, Tyrolean termination tool 2. In R. Treinen, editor, Rewriting Techniques and Applications, volume 5595 of Lecture Notes in Computer Science, pages 295-304. Springer Berlin / Heidelberg, 2009.

[22] P.Pan, T. Naduea, "Software-Defined Network (SDN) Problem Statement and Use Cases for Data Center Applications"ht*TP*s://tools.ietf.org/html/draft-pan-sdn-dc-problem-statement-and-use-cases-02 , Working paper IETF, 2012.

[23] S. Tison, " Tree Automata and Term reqrite systems Sophie Tison, LNCS 1833, Bachmair, L. Editor REWRITING TECHNIQUES AND APPS 11th International Conf RTA 2000, Nowrich UK, July 20000

[24] ht*TP*://www.openflowhub.org/blog/blog/2012/12/03/sdn-use-case-multipath-tcp-at-caltech-and-cern/,Visited Oct 10,2014