Near Optimal Service Function Path Instantiation in a Multi-Datacenter Environment

Ahmed M. Medhat, Giuseppe Carella
Department of Telecommunication Systems
Technical University Berlin
Berlin, Germany
a.hassan@mailbox.tu-berlin.de
giuseppe.a.carella@tu-berlin.de

Christian Lück, Marius-Iulian Corici,
Thomas Magedanz
Next Generation Network Infrastructures
Fraunhofer Institute FOKUS,
Berlin, Germany
{christian.lueck, marius-iulian.corici,
thomas.magedanz}@fokus.fraunhofer.de

Service Function Chaining (SFC) supports services/applications through linking an ordered list of service functions (such as firewalls, deep packet inspections and load balancers) in the network. Using Software Defined Networking (SDN) and Network Function Virtualization (NFV) provides a simpler and shorter SFC process. In order to provide high-availability on a multi-site SFC, it is required a flexible algorithm able to properly select Service Function instances while creating the Service Function Path (SFP). This paper presents a new service function selection algorithm providing a flexible tradeoff between the SFP distance and the loads per service function instances at the deployment stage. The flexible tradeoff provided in the algorithm can be adjusted by a parameter. This parameter value can differ among different types of SFCs based on the service/application (real time or non-real time) dedicated for its SFC. Matlab has been utilized for validating it with different scenarios. It is also compared to other existing algorithms such as, load balancing and shortest path. The results showed that the proposed algorithm could provide the best SFP end-to-end distance performance as the shortest path algorithm for the real time services/applications, and that will result a good Quality of Service (QoS) performance for real time services/applications. Furthermore, the proposed algorithm provides an acceptable load distribution over the available service functions instances better than shortest path algorithm and not far from and load balancing algorithms.

Keywords—Service Function Chaining; SDN; NFV; selection algorithms

I. Introduction

Network Function Virtualization (NFV) [1] is an arising telco operator initiative fostering the development of virtual network infrastructures by porting and further adapting Network Functions (NFs) to the specific cloud environment. NFV considers end-to-end Virtual Network Function Forwarding Graph (VNFFG) as a set of ordered chained NFs deployed on top of virtual resources. These NFs can be of any type, Firewall, Load Balancers and Deep Packet Inspections, and are typically integrant part of the network operator networks. Software Defined Networking (SDN) [2] plays an important role with NFV providing mechanisms for dynamically controlling this type of chains. Those mechanisms are currently addressed under the notion of Service Function Chaining (SFC) by IETF [3] and ONF [4]. SDN isolates the control and data planes and provides appropriate programming abstractions. It is used in SFC to facilitate traffic steering of flows across the specified Virtual NFs (VNFs) using appropriate techniques. At the same time there is an increasing interest in moving the data towards the user making use of Mobile Edge Computing (MEC) mechanisms introduced by ETSI [4]. With the introduction of datacenters on the edge of the network, the complexity of managing the lifecycle of network flows will be arising exponentially. Many network operators need to relocate and distribute their network functions across multi-datacenters to meet the locality, costs and latency required objectives[6]. ETSI [7] implemented the NFV infrastructure (NFVI) as a distributed set of NFVI nodes, deployed in different locations (NFVI Point of Presence), such as customer premises, network exchange points, and datacenters. VNFs can be dynamically located at the most appropriate NFVI PoP [7]. This distributed model of VNFs instantiation at different locations has been mentioned as Distributed NFV [8].

Deployment of multiple instances of the same VNF type across multi-cloud domains provides a challenge to select properly the VNF, as there are multiple instances of the same VNF type available in the network. Random selection of any VNF instance from the available ones, may lead to overload of some VNFs and consequently packets drop or do not meet the delay requirements for some services due to selecting far VNFs from the users. The challenge is increased with the extravagant increase in global IP traffic volumes (Fixed Internet, Managed IP and Mobile data). The Cisco's Visual Networking Index (VNI) mobile forecast predicts that over a five year period (2013 – 2018), global IP traffic will grow from 51,168 Petabytes per month to 131,553 Petabytes per month [9]. An appropriate VNF selection algorithm needs to consider this huge increase in data traffic without backsliding the quality and reliability of the provided services and without an increase in the network costs (CAPEX and OPEX).

Not so much work has been realized in the context of VNF selection, even though it can be seen as a specialized topic in the context of load balancing and high-availability. We categorize VNF selection algorithms into two classes. The first class is a uni-objective VNF selection approach, where the algorithm's target is to enhance only one objective, for instance providing load balancing among all VNFs. The second class is a multi-objective VNF selection algorithm as a combination of multiple objectives.

The proposed VNF selection algorithm considers the VNF loads, Application types (QoS Class) and the delay constraints. The dynamicity and flexibility of the proposed algorithm provides different ways of selection by changing some parameters´ values to meet the requested service and users requirements. For this work, we consider only the deployment phase of SFC. The proposed algorithm is validated and compared to other known algorithms to show the performance enhancement.

The rest of the paper is organized as follows. Section II provides the related work. Discussing how VNF selection works and introducing the proposed algorithm in Section III. Validate the proposed VNF selection algorithm and evaluate the performance results in Section IV. Finally, Section V summarizes the paper and discusses the future works.

II. Related Work

This section presents some background information about SFC architecture, SFC in multi-datacenters environment and load balancing mechanisms which could be considered as a use case of VNF selection algorithms in SFC.

A. SFC Architecture

In a high level view over the complete SFC architecture according to IETF SFC specification, there are three layers composing the SFC architecture. These layers are management plane, control plane and data plane. The management plane layer is known as the SFC orchestrator which is responsible for SFC management and SF instances management. The control plane layer is responsible for mapping SFC to specific Service Function Path (SFP), installing and administering forwarding rules on the data plane and adjusting the SFP based on SF instances status and overlay links [10].

The main SFC architecture's components in the data plane layer, includes classifier, Service Function (SF), SF Proxy and Service Function Forwarder (SFF). The classifier differentiates the traffic based on the requested application and other predefined conditions, and then it redirects it to the specified SFC by adding an SFC header containing a path ID into the flow packet headers. SFC is an ordered group of abstract SFs which must be applied to the traffic's packets elected as a result of classification. NFs or VNFs are named SF in the service function chaining architecture model by IETF. A SF is a function responsible for applying specific task on received packets and can be shared among multiple SFPs. A single SF can be implemented in multiple instances for scalability reasons, and those instances could also be placed in different locations. A SFF is responsible for forwarding the traffic to SFs or other SFFs according to the information carried in the packet's encapsulation based on the assigned SFP. IETF SFC does not specify a specific SFF, but the SFF must be able to insert Network Service Header [11] (SFC special header) to the traffic packets. SFP is the actual path (the exact SFFs/SFs) which the traffic's packets steer through it. It might be needed a SF proxy between the SFF and SFs as most of SFs do not understand the SFC packet headers. This SF proxy de-encapsulates the SFC headers from the packets when the traffic goes into the SF and encapsulates it again when it is coming out from the SF to the SFF [3].

Looking at traffic steering among SFs, it is possible to classify them into two different categories. The first category uses labels or special headers to steer the traffic using one classifier such as work presented in [11] and [12] . The second category does not use this label or special header but implement per-hop classifier to steer the traffic across the needed SFs such as work discussed in [13], [14] and [15]. There are many works discussing about orchestration and management of SFs and SFC, but these works are out of the scope of this paper as they are mainly focused on management layer functionalities, while the presented algorithm is more on the data plane layer.

B. SFC in Multi-datacenters Environment

In a large scale multi-datacenter environment usually long distances are separating datacenters from each other. In each datacenter sets of multiple SFs of various types are deployed. These sets of SFs (virtual or physical), hosted at one datacenter, may have equivalent sets hosted in other datacenters. In such deployments, traffic may cross multi-datacenters in order to pass through the SFs instances according to the selected SFP. A full distribution of equal SF instances over all involved datacenters is the use case this work is addressing. ETSI NFV ISG [16] provides a proof of concept that demonstrates an automatic deployment of VNFs across all available NFVI instances, automatic failover and reinstallation of failed VNFs. Moreover, it realizes redundancy and near-linear scalability by specialized middleware which provides state replication and synchronization services between VNF instances

There is also previous work in [17] providing optimal distribution of VNFs in multi-cloud environments. Providing SFC across multi-cloud domains is considered as a use case by IETF SFC working group [18], where SFCs steer across multi-datacenters and enable operators to deploy SFs in a flexible way. There is some work done before which provides SFC across multi-cloud environments such as Cloud4NFV [19], MIDAS [20], and the work presented in [21].

C. Load Balancing Algorithms

Load balancing is essential in service function chaining as it is stated in the requirements for service function chaining by IETF SFC WG [22]. SIMPLE [15] describes an approach for load balancing service instances. They formulate the NFs load balancing problem as a linear program. They choose as a specific load balancing objective to minimize the maximum NF load across the network. Authors in [23] differentiated load balancing algorithms, in the context of cloud computing, into four different classes, applying a generic classification methodology. The first class is the static environment that needs a prior knowledge about nodes and user requirements such as Round Robin, Opportunistic Load Balancing, Min-

Min and Max-Min [24] algorithms. The second class is the dynamic environment which doesn't need prior knowledge of nodes and it relies on run-time statistics such as Load Balance Min-Min (LBMM) algorithm [25]. The third class is centralized load balancing which all the decisions are taken by a single node, therefore it results an overhead on this node. The fourth class is distributed load balancing, which there are multiple nodes are monitoring cloud network and take decisions such as Honey Bee Behavior Inspired Load Balancing algorithm [26] and Ant Colony Optimization algorithm [27].

III. Service Function Selection Algorithm

One of the IETF SFC WG specifications' requirements is to allow deploying multiple instances of each SF type in the network for load-balancing, high availability, and failover needs [22]. The same type of SFs instances can be distributed over multiple locations for network offload, latency, reliability and availability reasons [6]. Many SFPs could be created for one SFC as a result of permutation of all available SFs instances in the SFC. For example as shown in figure 1, SFC 1 is defined as a set of ordered abstract SFs {SF1, SF3, SF4} and having three instances for each SF_z abstract (SF_{z1}, SF_{z2} and SF_{z3}) distributed in different locations. For each SF abstract in the SFC it is necessary to select one SF instance. In our model, one SFP is created at the end from all possible SFPs for one SFC. Any SF instance could be reused in multiple SFPs for different SFCs [22]. SF Selection algorithm is needed to choose the most appropriate SF instance available which meets the users' and services/applications' requirements. The proposed SF selection algorithm is applied only at the deployment phase of the SFC. SF selection algorithm is employed for constructing a SFP based on the SF involved in the chain. In our model, we assume that one only SFP is instantiated per SFC.

A. *Proposed SF Selection Algorithm*

SFs' loads are important issue that needs to be considered in SFC. When a SF instance is overloaded, it will increase the processing time of packets at the SF instance and it will result in an increasing number of dropped packets or high delay of transferring the packets. The high delay of transferring packets through the SFPs can be also as a result of selecting SFs away from each other, which will provide a long path between consecutive SFs in one SFP and consequently a long end-to-end path's distance between the source node (user) and the destination node (service/application) is produced.

The proposed algorithm's approach is to provide a trade-off between the SF's load and the delay. We need to decrease the end-to-end traffic delay especially for the real time services/applications without overloading SFs. An end-to-end shortest path (from source to destination node passing through the SFs instances as specified by SFP) is used to provide the lowest possible distance, which will consequentially result in a low delay. At the deployment phase of the SFs, when there is no traffic, all SFs' loads are zero as these SFs are used only by the SFC network. The SFs' loads are calculated based on the involvement of the SF instance in SFPs. For example, if a SF instance is used in two different SFPs then its load will be equal to two. For the use case used in this paper, there are multiple instances of the same SF type in different locations (datacenters). The distance between datacenters is considered as a constant delay time. Even if in a real scenario delay among datacenters is not a static parameter, this assumption has been introduced as in this work we are not considering the dynamicity aspects of the systems involved. However, the proposed algorithm and system could be easily extended for supporting also the dynamicity of the system. There are multiple instances of each SF type to be utilized during SFP construction. Each SF type with ID *x* has *Y* instances in the network, each SF instance has ID *y*. So there are many choices for each SF type in constructing multiple SFPs as a permutation of these multiple SFs instances based on the corresponding SFC. During SFP building, an index value is calculated based on equation (1) for all SF instances for each SF abstract in the corresponding SFC and then the lowest index is chosen to be the SF instance selected for SFP instantiation.

$$Index_{SF_{xy}} = \frac{(1-\alpha_{xy})\times L_{xy}}{Max(L_x)} + \frac{\alpha_{xy}\times P_{xy}}{Max(P_x)} \quad , where\ 0 \le \alpha \le 1 \qquad (1)$$

The Index of a SF instance for one SFP is noted as $Index_{SFxy}$ where L_{xy} is the load of the SF_{xy}, P_{xy} is the shortest path distance between the current node such as the Ingress Gateway (which the users are connected to it) or the SF instance and the next SF instance according to the SFP specified, and α_{xy} is a value between 0 and 1 to provide a flexible trade-off between SF's load and end-to-end delay. L_x represents all available SFs' instances loads for SF with type ID *x*, while P_x is all available path distances between the previous selected SF in the SFP (or the classifier node) and all candidates SFs instances with type ID *x*. The usage of Maximum value of L_x and P_x in calculating the Index value is to provide normalization in calculating the SF instances loads and Path distances which allows scalability in the number of SFCs in the network.

The Index value calculation is based on the characteristics of applications requested (Real Time (RT) App, Non-Real Time (NRT) App, or NRT with High priority (NRT HP)) which is considered while calculating the α_{xy} parameter. This parameter is a dynamic value and it is calculated by equation (2). It relies on the priority of the SFC depending on the type of Service/application (RT, NRT or NRT HP) specified for this SFC and the SFs instances load of other available SFs instances from the same type ID *x*. The highest priority (*HP*) value is provided for RT-Services (set to 5), the lowest value is provided for NRT-Services (set to 1) and for High Priority NRT-Services, the priority value (*SFCpr*) is set to 3.

$$\alpha_{xy} = \frac{\sum_{m=1}^{Y}(L_{xm}) - L_{xy}}{2\times\sum_{m=1}^{Y} L_{xm}} + \frac{SFCpr}{2\times(HP+1)} \qquad (2)$$

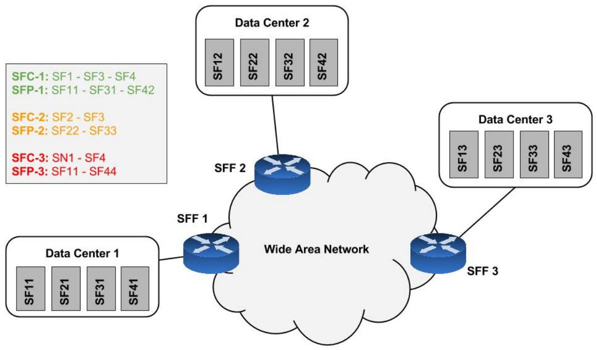

Fig. 1. VNFs Deployment in Multi-datacenters

The shortest path is calculated based on Dijkstra algorithm. The shortest path distance for the last SF in a SFP is calculated between the previous (SF or GW) node and the last SF instance selected in the SFP in addition to the distance between the last SF instance and the Egress Gateway (which servers/applications are connected to it). The load is calculated based on how many times each SF instance is used in all constructed SFPs. If the α_{xy} value calculation result is greater than 0.5, the results are the best in terms of end-to-end delay, therefore near optimal for real time services and applications. In case of other delay tolerant services, α_{xy} value calculation result will be lower than 0.5 and that will increase the load balancing consideration and maintain the shortest path at the same time.

Flow chart in figure 2 shows the proposed algorithm steps in selecting the SF instances. Firstly, for each SF instance *(SF_{xy})*, get the SF instance's *(SF_{xy})* load, calculate the shortest path between the previous node (GW or SF instance) and this SF instance *(SF_{xy})* and calculate the α_{xy} parameter value based on equation (2). Then, calculate the index value for each SF instance and select the SF instance with the lowest index value to be inserted in the SFP. Finally go to the next SF in the SFC until the last one and do the same calculations in order to select the SF instances to create the SFP.

B. Existing SF Selection Algorithms

OpenDaylight (ODL) [28] SFC Project provides four different known SF selection algorithms in the latest release of ODL (Lithium) such as Random, Round Robin (RR), Load Balancing (LB) and Shortest Path (SP).

Random SF Selection Algorithm is selecting the SF instance in a random way without any consideration to other factors such as load or path length. RR SF Selection Algorithm is based on selecting the SF instance in a defined order, which means if SF_{11} is used previously in a SFP then the next selection for SF1 abstract will be SF_{12}. Such algorithm considers only the order of SFs instances in selections and when they are all selected, it returns back to select the first one from the list of SFs instances.

LB SF Selection Algorithm is based on SF instances load. ODL SFC project pretend on monitoring the SF instances loads and select the lowest SF instance's load. Since it is a deployment phase, so there is no traffic which monitoring reads zero load at all SFs. Such algorithm could not be used for constructing SFPs; it should be used when there are traffics generated in the network in order to be able to monitor the loads of SF instances. In our implementation, the load is calculated based on how many times the SF instances are involved in the previous constructed SFPs for other chains.

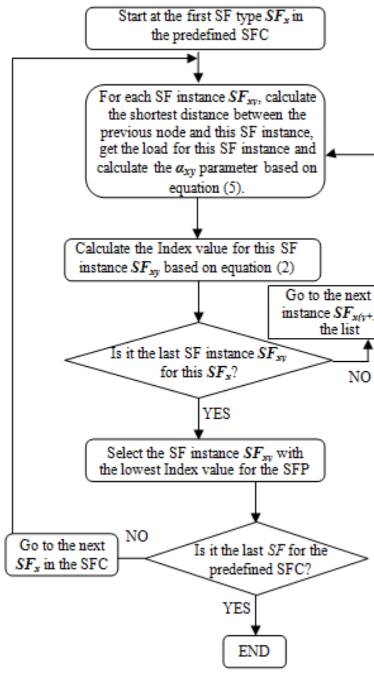
Start at the first SF type SF_x in the predefined SFC
For each SF instance SF_{xy}, calculate the shortest distance between the previous node and this SF instance, get the load for this SF instance and calculate the α_{xy} parameter based on equation (5).
Calculate the Index value for this SF instance SF_{xy} based on equation (2)
Go to the next SF instance $SF_{x(y+1)}$ in the list
Is it the last SF instance SF_{xy} for this SF_x?
NO
YES
Select the SF instance SF_{xy} with the lowest Index value for the SFP
Go to the next SF_x in the SFC
NO
Is it the last *SF* for the predefined SFC?
YES
END

Fig. 2. Flow chart for the proposed SF selection algorithm

SP SF Selection Algorithm is based on selecting the shortest distance path between the previous (GW or SF) node and the next SF instance in the SFP. SP's calculation is based on Dijkstra Algorithm. SP Algorithm considers only the distance between the SFs in the SFP without considering the loads of SFs instances. Also it does not consider the location of final destination (service/application).

IV. Validation and Evaluation

The proposed SF selection algorithm implementation and validation are discussed in the following sub-section A and pre-evaluation results are provided in sub-section B. The evaluation shows the efficiency of the proposed algorithm comparing to other existing algorithms.

A. Implementation and Validation of the proposed algorithm

The proposed algorithm has been implemented in Matlab and it has been evaluated comparing its results with the existing SF selection algorithms of ODL.

The topology of network was constructed as shown in figure 3. This scenario is applicable for Mobile Core Networks. There are 17 nodes in the network (2 GWs act as classifiers, 3 SFFs and 12 SFs instances). There are 4 SFs instances from different types at each datacenter, and they are all controlled by the same SFF (also one per datacenter). The distance between nodes are based on the delay time of the links and is in milliseconds (ms). The distance between the users and Ingress GW passing through the Mobile Network is neglected in our calculations. The distance between the GW and nearest connected SFF is equal to 2 ms and the distance between each SF instance and its corresponding SFF is equal to 1 ms. The distance between SFF1 and SFF2 is equal to the distance between SFF2 and SFF3 equals to 6 ms. The distance between SFF1 and SFF3 is equal to 8 ms.

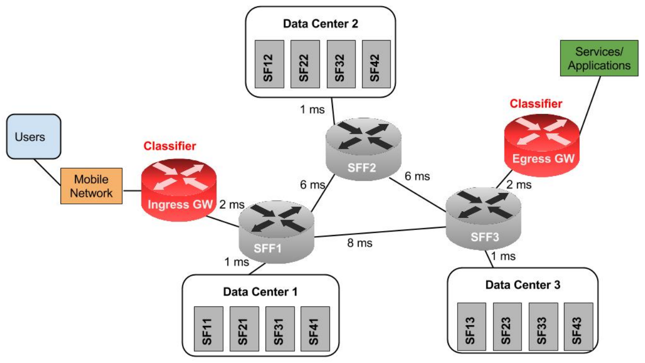

Fig. 3. Topology for testing and validation

B. *Performance Evaluation of the Proposed SF Selection Algorithm*

The proposed algorithm is evaluated through testing it over the topology shown in figure 3. Also its performance is compared to, SP and LB SF selection algorithms. After constructing the six SFPs based on the six predefined SFCs shown in table 1 using the SF selection algorithms (the proposed, LB, and SP algorithms), the end-to-end path's delay for each SFP is calculated and the load of each SF instance is also computed.

TABLE I. SERVICE FUNCTION CHAINS USED IN THE TESTING SCENARIO

SFC ID	Service/ Application Type	1st SF in the Chain	2nd SF in the Chain	3rd SF in the Chain
SFC 1	NRT HP	SF1	SF2	SF3
SFC 2	RT	SF3	SF4	SF1
SFC 3	NRT	SF2	SF3	SF4
SFC 4	NRT HP	SF4	SF1	SF3
SFC 5	RT	SF3	SF4	SF2
SFC 6	NRT	SF4	SF2	SF1

Fig. 4 shows the results of end-to-end path's delay for each SFP using the proposed SF selection algorithm and the other two existing SF Selection Algorithms (SP and LB). The end-to-end distance covered for each SFP is starting from the Ingress gateway then SFs instances according to the constructed SFPs and pass by the egress gateway then finally reaches the service/application node. The load per each SF instance is presented in fig. 5, after instantiating the SFPs using the proposed SF selection algorithm and the other existing SF selection algorithms. The SF instances in figure 5 with IDs {1, 5, and 9} are considered to have the same service function type SF1. The SF instances with IDs {2, 6, and 10} belong to the same service function type SF2. Service function type SF3 has SF instances with IDs {3, 7, and 11} and Service function type SF4 has SF instances with IDs {4, 8, and 12}.

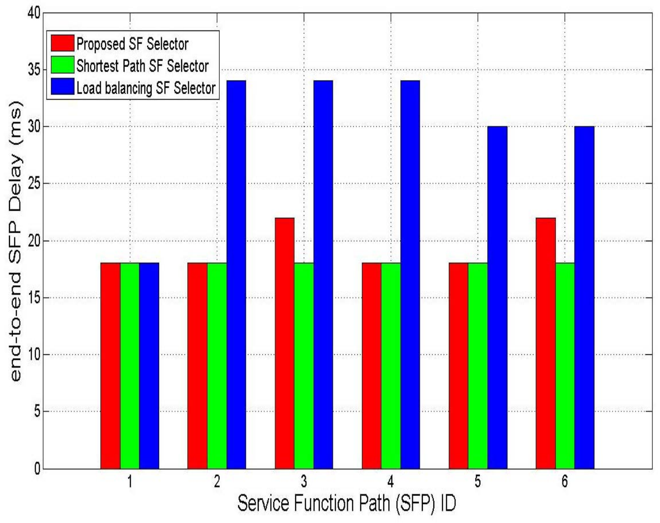

Fig.4. End-to-end Delay for constructed SFPs

As shown in figure 4, the Load Balancing SF Selection algorithm is the longest end-to-end delay for most of given SFPs compared to the proposed and Shortest Path algorithms. This increase in delay will definitely affects on the real time services/application (SFP 2 and SFP 5). On the other hand, the shortest path algorithm and the proposed algorithm provide shortest delay especially for the RT services/applications (SFP 2 and SFP 5) and also for the NRT services/applications with high priority (SFP 1 and SFP 4). The proposed algorithm also provides better results in terms of delay than the load balancing algorithm for the SFPs (SFP 3 and SFP 6) specified for NRT services/applications.

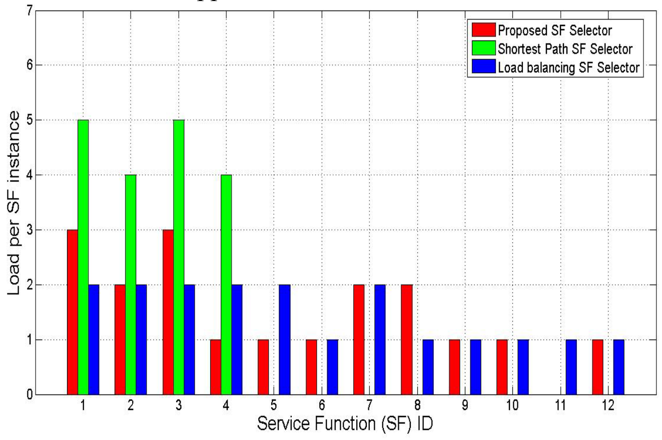

Fig. 5. SF instances' loads after constructing the SFPs

Load balancing algorithm shows the best results in SF instances' loads as shown in figure 5. The loads are very well distributed among the SF instances. Shortest Path algorithm's results show a very high load in the nearest SF instances (in Datacenter 1) to the Ingress GW and the rest are zeros. These increased loads in some SF instances may lead to a delay or dropped packets and that is not acceptable for real time Services/applications. The proposed algorithm shows an acceptable load distribution among the available SF instances.

The proposed SF selection algorithm in this paper shows an overall very good performance. It provides the shortest end-to-end distances for the RT services/applications that will support decreasing the traffic delay. It also does not neglect the SF instances' loads, in which it provides acceptable balanced loads among the SFs instances. In comparison with the two other SF Selection algorithms, our proposed SF selection algorithm shows that it has the best performance in both SF instances' loads distribution and the end-to-end delay of SFPs especially for the RT and NRT HP service/applications. The algorithm needs to be evaluated in a real environment in order to prove its preliminary results in this paper and relying on live monitored SF instances' loads and links' delays values.

V. Conclusion and Future Works

This paper presents a new SF selection algorithm employed for constructing SFPs in a multi-datacenter environment SFC deployment. The algorithm provides a tradeoff between the end-to-end path's distance and the SFs instances' loads. Using a dynamic parameter, it provides the flexibility in the way of SF selection based on the requested services/applications requirements and the loads of other SF instances. The algorithm is validated and compared with other known algorithms. It shows near optimal results in both the SFs Instances' loads distribution and providing shortest path delay for the SFPs dedicated for RT services/application and also NRT services/applications with high priority.

As a future work, the proposed algorithm will be implemented in the ODL SFC Project in order to evaluate and prove its benefits on real networking environment. Using a real system, we will be able to monitor the actual SF instances' loads and links delays and apply the proposed algorithm also on the runtime phase of the SFC lifecycle. In this way, it will be possible to instantiate multiple SFPs for each SFC in order to provide dynamic SFP on runtime to avoid congestion and SFs instances overloads.

Acknowledgment

This Project has been funded with support of the European Commission. This publication reflects the view only of the authors, and the Commission can't be held responsible for any use which may be made of the information contained therein.

References

[1] NFV White Paper, "Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action.", Issue 1, Oct 2012.

[2] D. Kreutz, F. M. V. Ramos, et. al., "Software-Defined Networking: A Comprehensive Survey", in Proceedings of the IEEE, vol. 103, no. 1,p. 63, 2015.

[3] J. Halpern, and C. Pignataro, " Service Function Chaining (SFC) Architecture", IETF SFC WG, March, 2015.

[4] ONF White Paper, " L4-L7 SFC Solution Architecture", June 2015.

[5] ETSI MEC, " Mobile-Edge Computing", Sept. 2014. Introductory Technical White Paper (issue 1). https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf

[6] Collaborative White Paper between Alcatel-Lucent and Telefonica, ¨Why distribution matters in NFV¨, August 2014.

[7] ETSI," Network Functions Virtualisation (NFV); Architectural Framework¨, ETSI GS NFV 002 V1.2.1, December, 2014

[8] ETSI, "Network functions virtualisation use cases," ETSI GS NFV 001 v1.1.1, October 2013.

[9] Cisco Visual Networking Index: Forecast and Methodology, 2013–2018.http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html

[10] H.Li, Q.Wu, et. al., "Service Function Chaining (SFC) control plane components and requirements", IETF SFC WG, June 2015.

[11] P. Quinn and J. Guichard, "Service Function Chaining: Creating a Service Plane via Network Service Headers", in IEEE Computer Journal, vol.47, issue 11, pp 38-44, Nov. 2014.

[12] S. Fayazbakhsh, V. Sekar, M. Yu and J. Mogul, "FlowTags: Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions", in the second ACM SIGCOMM workshop on hot topics in software defined networking (HotSDN), Hong Kong, August 2013.

[13] Y. Zhang, N. Beheshti, L. Beliveau et. al, "StEERING: A Software-Defined Networking for Inline Service Chaining", in 21st IEEE International Conference on Network Protocols (ICNP), Goettingen, Oct. 2013.

[14] J. Blendin , J. Ruckert, N. Leymann et. al., "Position Paper: Software-Defined Network Service Chaining", in Third European Workshop on Software Defined Networks (EWSDN), Budapest, Sept. 2014.

[15] Z. Qazi, C. Tu, L. Chiang et. al., "SIMPLE-fying Middlebox Policy Enforcement using SDN", in ACM Conference on SIGCOMM, New York, 2013.

[16] ETSI NFV ISG, "Demonstration of multi-location, scalable, stateful Virtual Network Function", December 2014

[17] D. Bhamare, R.Jain et. al, "Multi-Cloud Distribution of Virtual Functions and Dynamic Service Deployment: OpenADN Perspective", in IEEE International Cloud Engineering (IC2E), USA, March 2015.

[18] S. Kumar, M. Tufail et. al., "Service Function Chaining Use Cases In Data Centers", IETF SFC WG, January 2015

[19] J. Soares, C. Goncalves et. al, "Toward a Telco Cloud Environment for Service Functions", in IEEE Communication Magazine, Vol. 53 issue 2, p.p. 98-106, February 2015.

[20] A. Abujoda and P. Papadimitriou, "MIDAS: Middlebox Discovery and Selection for On-Path Flow Processing", in IEEE International Conference on Communication Systems and Networks (IEEE COMSNETS), Bangalore, India, January 2015.

[21] F. Callegati, W. Cerroni, C. Contoli and G. Santandrea, "Dynamic Chaining of Virtual Network Functions in Cloud-Based Edge Networks", in IEEE Conference on Network Softwarization (NetSoft), London, April 2015.

[22] M. Boucadair, C. Jacquenet, Y. Jiang and et. al. "Requirements for Service Function Chaining", IETF Draft, February 13, 2015.

[23] M. Patel and R. Mehta, "A comparative study of heuristic load balancing in cloud environment", in international Journal of Advance Engineering and Research Development, Vol. 2, Issue 1, January 2015.

[24] S. Wang, K. Yan et. al, "Towards a load balancing in a three level cloud computing network", in 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Vol. 1, Chengdu, July 2010.

[25] T. Kokilavani, and G. Amalarethinam, "Load balanced Min-Min Algorithm for static meta-task scheduling in grid computing", in International Journal of Computer Applications (0975-8887), Vol. 20 - No.2, April 2011.

[26] D. Babu, L.D & P. Krishnab, "Honey bee behavior inspired load balancing of tasks in cloud computing environments", in Applied Soft Computing, Vol. 13, Issue 5, May 2013.

[27] M. Dorigo and C. Blum, "Ant colony optimization theory: A survey", in Theoretical Computer Science, Vol. 344, Issues 2-3, pp. 243-278, August 2005.

[28] OpenDaylight.https://opendaylight.org/