Optimal fog services placement in SDN IoT network using Random Neural Networks and Cognitive Network Map *

Piotr Fröhlich[1][0000−0002−4854−4256] and Erol Gelenbe[1][0000−0001−9688−2201]

Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Gliwice, Poland
pfrohlich@iitis.pl,
e.gelenbe@imperial.ac.uk

Abstract. Due to a massive increase in the number of IoT devices and the number of cloud-based services a crucial task arises of optimally placing (both topologically and resource-wise) services in the network so that no of the clients will be victimized and all of them will receive the best possible time of response. Also - there must be a balance not to instantiate a service on every possible machine - which would take too many resources. The task which must be solved is an optimization of parameters such as QoS between service and client, equality of clients and usage of resources. Using the SDN - which is designed to answer some of the problems posed in this section such as QoS and knowledge about the topology of the whole network and newly connected clients - is a gateway to better-adapted service management. Machine learning provides less stiff rules to follow and more intelligent behavior of the manager.

Keywords: Random Neural Network · Reinforcement Learning· Artificial Intelligence· IoT· SDN· Fog Computing· Cloud Computing

1 Introduction

Currently, a lot of online services are based on cloud or fog management - by certain calculations *82%* of the workload will reside within the cloud by 2020 [17]. It poses a serious problem of the management of used resources and the location on which such service should be optimally placed. This problem seems to be more and more serious, especially when the number of IoT devices is growing rapidly (as much as 5.8 Billion IoT endpoints [12] by 2020) and consequently - the number of queries to servers supporting these devices is growing. It gets more and more crucial for operators and providers to be able to deploy services in the best possible place. Users expect efficient service, which means low response time

* This research has been supported by the EC H2020-IOT-2016-2017 (H2020-IOT-2017) Program under Grant Agreement 780139 for the SerIoT Research and Innovation Action.

and fair service, i.e. similar service time regardless of the customer's location. On the other hand, for the infrastructure operator, it is important to optimize the use of computing and network resources - to maximize the number of supported users (which is associated with the number of launched services) and minimize the consumed energy. The sage of *SDN* [4] gives this solution a lot of information that is otherwise impossible or hard to obtain in regular networks. The fact that *SDN* is used speeds up the setup of the required application. What's more *SDN* is often used in Cloud and Fog networks [14, 13]. Also Cognitive Network Map which is fed by Cognitive Packets [8][10] with information about the current state of the network (e.g. *QoS*-wise) and stores it within the controller. The whole network is constantly monitored with the use of real, cognitive packets using real interfaces to travel through. It provides the Cognitive Network Map (*CNM*) with actual measurements of *QoS* parameters.

Today's development of information processing technology allows the use of one of the many heuristic methods of solving optimization problems. Artificial intelligence systems [16, 11] are particularly promising. In this paper, we decided to test the usefulness of Random Neural Networks [7, 6] for tasks related to supporting the work of a distributed system operator of Fog-for-*IoT* type systems.

Using the *RNN* based algorithm with the *RL* algorithm [9] for constantly deciding on optimal placement of a given service provides end-user with stable *QoS* parameters and provider with close to optimal resource consumption. The work is based on the work carried out within the framework of the SerIoT project [5][2], which includes the use of the Fog subsystem for the installation of services for *IoT* equipment. The choice of *RNN* network was also motivated by the use of the type neural networks in this project[15].

In the following section all submodules and algorithms will be described. Then the actual solution will be presented, followed by a description of testbed and experiments. Everything will be concluded in the last section with an emphasis on possible improvements.

2 Description of used algorithms and submodules

2.1 Description of Random Neural Network

The topology of the neural network used in the decision process is a fully recurrent equivalent of the topology of the real network (see Fig.1.). For every service that can be deployed such a network is created. Neuron(s) with a potential

$$P \in (P_{MAX} - TR, P_{MAX}) \tag{1}$$

where P_{MAX} is a maximal potential in the whole neural network and

$$TR = 1 - (CPU_{USED}/CPU_{MAX}) \tag{2}$$

are considered winning neurons. On every forwarder which corresponds to a winning neuron a service is then instantiated - the process of spawning a service

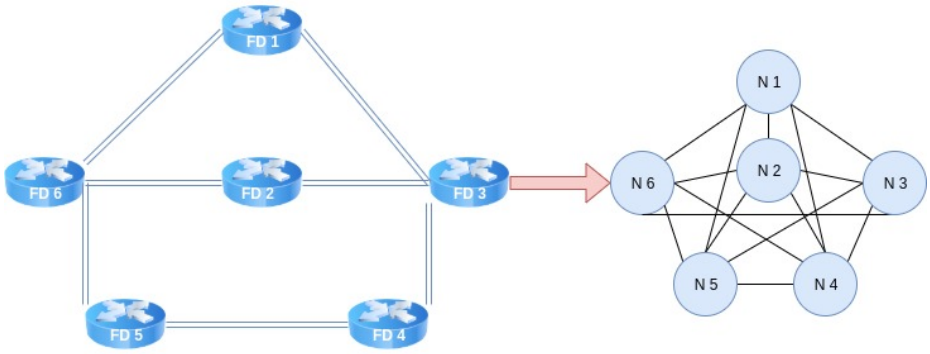

Fig. 1. The topology of the real network and the Random Neural Network corresponding to it.

is described in subsection *2.3*. Once a decision of a neuron takes place it is constantly scored based on that decision - reward, R, which a single neuron receives can be described in the equation below. Every part of this equation is explained in respective subsections.α, β, γ are parameters that represent the importance of a given part of the equation.

$$G(service) = \alpha * S(service) + \beta * \sum_{i=0}^{i=C} D(p_i) + \gamma * W(service) \tag{3}$$

$$R = \frac{1}{G} \tag{4}$$

Function S (service). This function is a standard deviation estimator (s) calculated for every client connected to a given service at a given time. This part of the goal function corresponds to the equality of clients. The higher the value the bigger the punishment.

$$s = \sqrt{\frac{\sum_{i=0}^{i=N}(p_i - \overline{p})^2}{N-1}} \tag{5}$$

Function corresponding to the β parameter. This function is an averaged QoS parameter of every client connected to a given service. QoS parameters are taken from the *Cognitive Network Map* which is described in subsection *2.2.* This part of the goal function corresponds with the overall QoS score of the placement of a given service. Note that C is a number of clients connected to a given service.

Function W(service). This function returns the workload of a given service. The higher the workload of a single service the higher is the need for instantiating another service.

As was stated the task of Random Neural Network is to minimize goal function using the Reinforcement Learning algorithm. Firstly the learning algorithm will update a given value:

$$T_l = \delta * T_{l-1} + (1 - \delta) * R_l, 0 < \delta < 1 \tag{6}$$

Where δ is a responsiveness parameter describing how important historical values of rewards are. Setting it to a very high value will prevent the neural network from taking hasty decisions. The R_l is calculated as described in (2) so every time the *CNM* is updated, new reward R_l will be calculated and neurons' weights will be recalculated as described below. As stated in the [3] *RNN*'s learning algorithm works rewarding neurons that were able to produce better reward value, R_l, than the history-aware value T_l calculated in (6). Then all positive weights leading towards that neuron are increased and negative weights leading from that neuron to the others are also increased. The *Reinforcement Learning* algorithm is described in [3, 9](7-12).

$$If \quad R_l >= T_{l-1} \quad then \quad for \quad j \neq k \tag{7}$$

$$\forall i \neq k, \quad W^+_{ik} \leftarrow W^+_{ik} + R_l, \quad W^-_{ij} \leftarrow W^-_{ij} + R_l \tag{8}$$

$$If \quad R_l < T_{l-1} \quad then \quad for \quad j \neq k \tag{9}$$

$$\forall i \neq k, \quad W^-_{ik} \leftarrow W^-_{ik} + R_l, \quad W^+_{ij} \leftarrow W^+_{ij} + R_l \tag{10}$$

After the normalization of weights, preventing weights from constantly increasing or decreasing, te reevaluation of the potential q_i takes place as follows:

$$q_i = \frac{\sum_{j=1}^{j=N} q_i * W^+_{ji}}{r_i + \sum_{j=1}^{j=N} q_i * W^-_{ji}} \tag{11}$$

and r_i is known as 'total firing rate'[3]:

$$r_i = \sum_{j=1}^{j=N} [W^+_{ij} + W^-_{ij}] \tag{12}$$

2.2 Description of Cognitive Network Map and Cognitive Packets.

Cognitive Packets mechanism provides a way to gather information regarding the physical status of the network. Every flow set by the controller in forwarders is being monitored with a certain frequency, *f*, by sending a Cognitive Packet through the given flow. All forwarders then add required information stored within themselves such as delay, packet loss, etc. Those packets are being sent to monitor every flow status. Assume network such as described in Fig.2. For

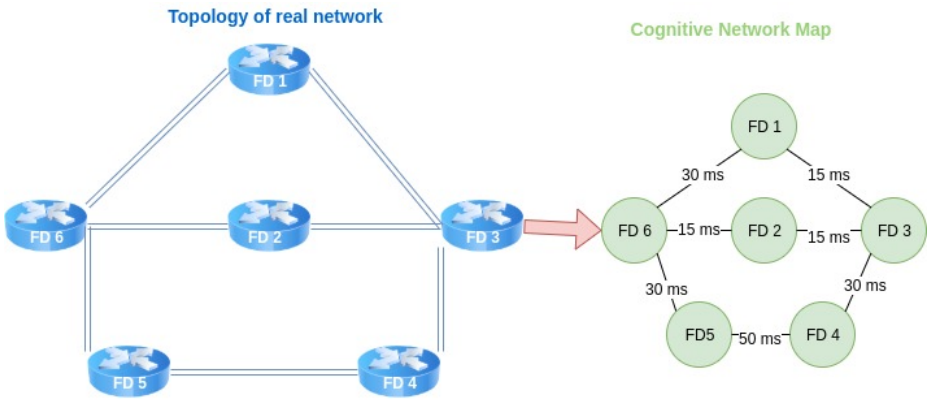

Fig. 2. The topology of the real network and the Cognitive Network Map for delay measurements only.

every flow installed by a controller - e.g. flow leading from the client connected to the *FD 1* to the client connected to the *FD 4*. Once every defined frequency f a packet is sent via this flow to monitor its state. Then, after receiving the data, it's sent to the controller for further processing. Using data collected by Cognitive Packets (*CP*) [8, 10] which are collected from real interfaces of a real network the Cognitive Network Map (Fig.2.) is created. To avoid being dependable on Cognitive Packets alone in providing data to the Random Neural Network a data aggregator is introduced to store necessary data.

In the described approach data provided by *CP* is averaged using the time frame of T. In time t all packets which arrived in an interval (t-T, t) are taken into account in creating the *CNM*, thus giving a solution an insight into the history. When Cognitive Packets are not reaching the controller (let's say on flow assumed previously) in a given time interval (t-T, t) links which are in the assumed flow begin to deteriorate at the given rate R - Fig.3.

2.3 Description of the service placement algorithm.

The service placement algorithm can be divided into two stages - the setup stage and the online stage. They will be covered in respective sections. The algorithm is working under two assumptions - firstly it starts its work with the stable *CNM* map. Secondly that controller knows every client connected to the network at any given moment. Using indications of the *RNN* which returns a number of maximally excited neurons corresponding with a machine on which a service can be instantiated. Working under the assumption that *CNM* is stable (meaning links are being monitored and *CNM* is an equivalent of the real network state) the first stage of the algorithm can take place.

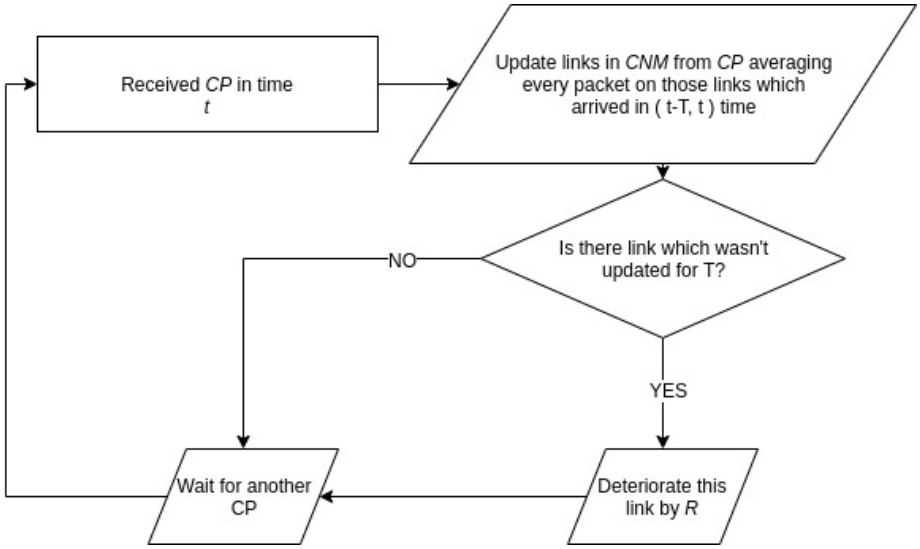

Fig. 3. Diagram showing the *CNM* creation algorithm.

Setup stage of the algorithm. After creating the *RNN* for a given service as described in subsection *2.1* algorithm proceeds with deciding on initial placement and number of services. Simulation is run, based on *CNM* data - every possible neuron is considered a winner and punished or awarded for it I number of times. After this phase, there are $S = \{s_0, s_1, ..., s_n\}$, N active services running in the network.

2.4 Online stage of the algorithm.

After the setup stage of the algorithm there are N running services and C connected clients. Every time the *CNM* is updated the RNN is also updated. If there are differences between sets $S_{t-1} = \{s_0, s_1, ..., s_n\}$, and $S_t = \{s_0, s_1, ..., s_m\}$ - winning services and winning neurons had changed, then all services:

$$S_{on} = S_t / S_{t-1} \tag{13}$$

$$S_{off} = S_{t-1} / S_t \tag{14}$$

are queued for starting and stopping. AfterΔ time, where Δ is proportional to δ that is the number of decisions of the *RNN* with the same tendency, all of S_{on} services will be started and all of S_{off} will be stopped.

3 Description of implementation, requirements and limitations

The use of the *SDN* made it possible to develop an application in a high-level language. As a controller *ONOS* [1] - an open-source solution - was used. It

provided a base for installing flow rules, routing, topology and host information. The controller is responsible for making changes in services placement - transparent for every client, store the *CNM* and *RNNs*. The whole *RNN* plugin is run as a thread inside of the controller.

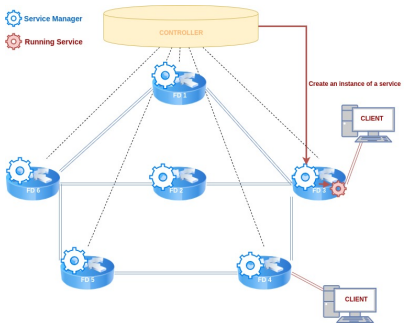

Fig. 4. Real topology used to deploy the presented solution.

Services must also be run on machines available for the service spawning - to know when and what service should be run. Therefore an independent, management network (Fig.4., dotted line) must be provided between machines running services and the controller. Those machines are marked as Service Manager (blue cog) and are in fact a REST interface daemons needed for communication between the controller and spawners of new services. Assume that a service must be instantiated on *FD 3*.

A REST request is sent to the *Service Manager* on that device which then instantiates a real service. From the client's point of view moving of the service is not visible due to masking flow rules installed by the controller. A client must only know an IP address and a port on which service S will be available all the time (regardless of placement) - rest is taken care of by the controller and the service manager. Regarding the requirements - a mentioned earlier management network directly connecting machines for spawning services and the controller is required. The *SDN* based switches are required - in this implementation both hardware *SDN* switches and *Open V Switch* based machines were used.

Since one of the assumptions made for the service placement algorithm is that the controller knows every connected host at all times - multiple clients connected to a router with the single IP address will be treated as one client (and will influence the balance of the algorithm). The network must also support Cognitive Packets - have nodes or clients running Cognitive Packets Managers all the time.

4 Description of testbeds and experiments

4.1 Description of testbeds

Testbeds used to conduct experiments can be divided into two groups - virtualized and real. For virtualized testbed setup a *mininet* environment was used. It provided required insight into large networks that are hard to create in the real world and it was easier to use as a performance testing tool. From now on it'll be referenced as the virtual testbed. The second testbed consists of real devices. Both *Switch 1* and *Switch 2* are providing both links in the abstract topology and the management network.

Machines marked with red colour are configured as *SDN* switches, *ARM*-based raspberry pi 3B+ devices. Those devices are running *OpenVSwitch* in version *2.3.0.* Other machines are (despite mentioned *switches 1* and *2*) regular desktop *PC*'s with *Ubuntu 18.04 LTS* running on them. Machine marked as *SerCon* is a machine on which the controller runs. In Fig.5. there is an abstract topology used during experiments conducted on real devices. This testbed will be referenced as the real testbed from now on.

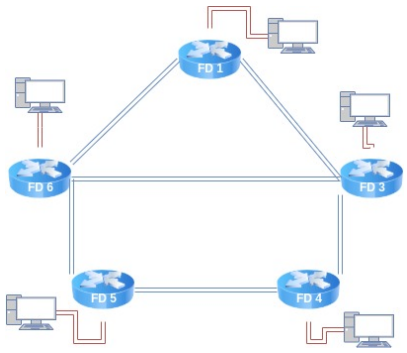

Fig. 5. Abstract topology used during experiments.

Several experiments were done to confirm that the solution is behaving correctly. To measure that every host connected to the network is treated equally a standard deviation of time of processing a request of every client was measured. Average processing time for all clients has been also measured as well as the average resource consumption. Described experiments were conducted both on the virtual and the real testbed.

4.2 Description of experiments

In experiments a network consisting of N nodes was created and C clients were randomly connected to those nodes. After T time a network state was captured and stored. Using the information in the *CNM* the solution based on the *RNN*

was able to provide optimal or suboptimal placements of services. Those experiments were run in the virtual testbed due to the ease of creating networks. Since packets using this virtual testbed are not transported by the real medium another set of experiments was needed. On the real testbed, a real value of time of the response, variation and resource consumption was measured. Note that all experiments conducted on real testbed were real services running on real machines. What's more - all traffic was transported via real medium - what's caused white noise.

5 Results

All experiments described in the previous section were conducted as described. Using both the *real* and the *virtual* testbed helped determine that this solution provides the network with optimal or suboptimal placements of services based on parameters such as the *QoS*, the *equality* of clients and *resource* usage. It also provided an insight into larger networks as well as real networks with real transportation media (carrying and implying a lot of white noise). All charts (Fig.6., Fig.7., Fig.8.) are charts taken from the real testbed from experiments carried out on the topology from Fig.5. with $C = 10$ and every forwarder ready to start a service at any time. The used service was a simple *TCP* based server which took N parameters and returned M parameters for the client. Both N and M were randomized but both of them were proportional to the *time of response*

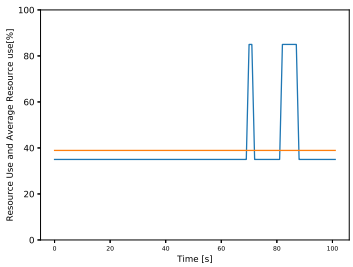

Fig. 6. Plot of use of percentage of used resourced against time.

Next experiments were conducted on the virtual testbed (Fig.9., Fig.10) with the use of bigger networks. Note that this isn't simulation - this is a virtualized solution with real systems, interfaces and services running in one environment (without the influence of the medium of transportation). All experiments were run for *100* [s], on topologies sized [*5, 10 , 15*] forwarders. Placement of the clients was randomized but clipped to a maximum of *2* connected hosts to one forwarder. The number of clients is equal to the number of forwarders to assure a stable condition of the network.

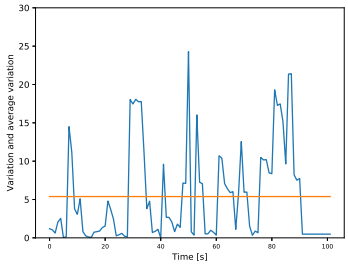

Fig. 7. The plot of the variation of the response time of every client against time. The orange line shows average variation through the whole experiment.

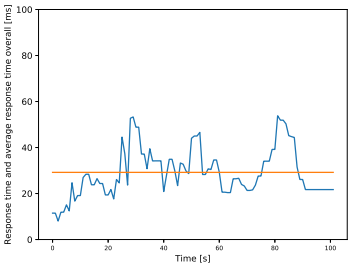

Fig. 8. The plot of the averaged response time for every client against time. The orange line shows average response time through the whole experiments

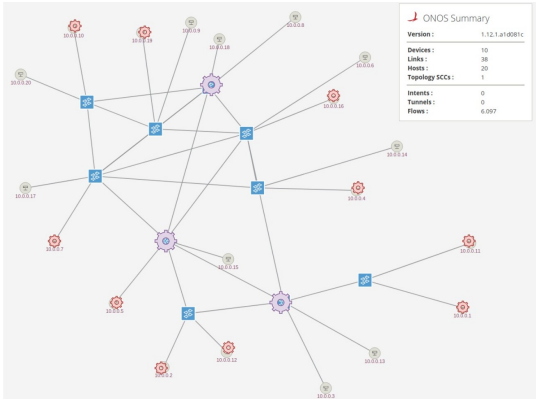

Fig. 9. The diagram showing the result of the algorithm. The red cog is a client, the violet cog is a running service.

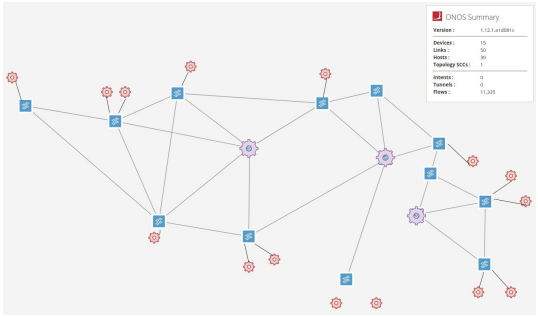

Fig. 10. The diagram showing the result of the algorithm. The red cog is a client, the violet cog is a running service.

6 Conclusion

The result of experiments that the *Random Neural Network* is a suitable solution for the service placement in the network. It provides means for optimization of required parameters and as a result best possible services for clients. Using the *CNM* provided the *RNN* with a stable source of data on the state of the network - as a result the service placement algorithm is also aware of a deterioration of the link quality, new clients and their location, etc. Combining decisions provided from the *RNN* and the data from the *CNM* resulted in a self-aware service placement algorithm that corresponds to changes in the network providing an optimal placement of services.

What's more, experiments concluded that this is a solution that can be used both in small-grade and large-grade networks. The usage of the *SDN* provides a way to mask a service replacement for clients (making it transparent to clients at any given time).

7 Acknowledgements

The author is grateful for the financial support of the H2020-IOT-2016-2017 Program under Grant Agreement 780139 for the SerIoT Research and Innovation Action.

References

1. Home page of onosproject - open source SDN controller. https://onosproject.org (2020), [Online; Last accessed 13 Mar 2020]
2. Home page of SerIoT project. https://seriot-project.eu (2020), [Online; Last accessed 13 Mar 2020]
3. Basterrech, S., Rubino, G.: A tutorial about random neural networks in supervised learning. ArXiv **abs/1609.04846** (2016)

4. Bera, S., Misra, S., Vasilakos, A.V.: Software-defined networking for internet of things: A survey. IEEE Internet of Things Journal **4**(6), 1994–2008 (Dec 2017). https://doi.org/10.1109/JIOT.2017.2746186
5. Domańska J., Nowak M., N.S.C.T.: European cybersecurity research and the seriot project. Communications in Computer and Information Science **935**, 166–173 (2018). https://doi.org/https://doi.org/10.1007/978-3-030-00840-6$_1$9
6. Gelenbe, E., Hussain, K.F.: Learning in the multiple class random neural network. IEEE Transactions on Neural Networks **13**(6), 1257–1267 (Nov 2002). https://doi.org/10.1109/TNN.2002.804228
7. Gelenbe, E., Koubi, V., Pekergin, F.: Dynamical random neural network approach to the traveling salesman problem. In: Proceedings of IEEE Systems Man and Cybernetics Conference - SMC. vol. 2, pp. 630–635 vol.2 (Oct 1993). https://doi.org/10.1109/ICSMC.1993.384945
8. Gelenbe, E., Zhiguang Xu, Seref, E.: Cognitive packet networks. In: Proceedings 11th International Conference on Tools with Artificial Intelligence. pp. 47–54 (Nov 1999). https://doi.org/10.1109/TAI.1999.809765
9. Gelenbe, E.: Learning in the recurrent random neural network. Neural Computation **5**, 154–164 (01 1993). https://doi.org/10.1162/neco.1993.5.1.154
10. Gelenbe, E.: Cognitive Packet Patent. https://patents.google.com/patent/US6804201B1/en (2020), [Online; Last accessed 20 Nov 2019]
11. Gupta, L., Samaka, M., Jain, R., Erbad, A., Bhamare, D., Metz, C.: Colap: A predictive framework for service function chain placement in a multi-cloud environment. In: 2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC). pp. 1–9 (Jan 2017). https://doi.org/10.1109/CCWC.2017.7868377
12. Inc., G.: IoT endpoint number. https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io (2020), [Online; Last accessed 20 Nov 2019]
13. Levin, A., Barabash, K., Ben-Itzhak, Y., Guenender, S., Schour, L.: Networking architecture for seamless cloud interoperability. In: 2015 IEEE 8th International Conference on Cloud Computing. pp. 1021–1024 (June 2015). https://doi.org/10.1109/CLOUD.2015.141
14. Mambretti, J., Chen, J., Yeh, F.: Next generation clouds, the chameleon cloud testbed, and software defined networking (sdn). In: 2015 International Conference on Cloud Computing Research and Innovation (ICCCRI). pp. 73–79 (Oct 2015). https://doi.org/10.1109/ICCCRI.2015.10
15. Nowak, M., Nowak, S., Domańska, J.: Cognitive routing for improvement of iot security (04 2019). https://doi.org/10.13140/RG.2.2.28667.36648
16. Ooi, B.Y., Chan, H.Y., Cheah, Y.N.: Dynamic service placement and replication framework to enhance service availability using team formation algorithm. Journal of Systems and Software **85**(9), 2048 – 2062 (2012). https://doi.org/https://doi.org/10.1016/j.jss.2012.02.010, selected papers from the 2011 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA 2011)
17. Radoslav Ch., T.: Cloud Computing Statistics 2020. https://techjury.net/stats-about/cloud-computing/gref (2020), [Online; Last accessed 20 Nov 2019]