Smart SDN Management of Fog Services

Piotr Fröhlich, Erol Gelenbe, *Fellow, IEEE* and Mateusz P. Nowak
Institute of Theoretical & Applied Informatics (IITIS-PAN)
Polish Academy of Sciences
ul. Baltycka 5, 44100 Gliwice, Poland
{pfrohlich,seg,mateusz}@iitis.pl

***Abstract*—A Smart Service Manager is proposed to direct user requests (such as those coming from IoT devices) at the edge towards appropriate servers where the requested services can be satisfied. Services are housed at different Fog locations, and the system is subject to variations in workload. The approach is based on using a Software Defined Network (SDN) controller as the means to take decisions with measurement data based machine learning that uses Reinforcement Learning to make the best choices. The system we have developed is illustrated with experimental results on a test-bed with time-varying loads that confirm its ability to adapt to significant changes in system load and preserve the users' Quality of Service.**

***Index Terms*—QoS, Edge and Fog Computing, IoT, Software Defined Networks, Machine Learning, Cognitive Packet Network, Random Neural Networks**

I. Introduction

Fog computing extends the Cloud [1] to allow edge devices to take over substantial computation, storage and networking, to facilitate the operation of services between edge devices and Cloud data centers [2]. It is particularly suited to manage services and tasks in the Internet of Things (IoT) [3]–[6].

Service virtualization is a characteristic of many computing platforms, and the Fog infrastructure offers computing nodes that run virtualized services that satisfy client requests. Thus the distribution or location of Fog nodes/servers in a network, and the placement of services on Fog servers are key issues. Since most networks may have a large variability of workload over time, the configuration of Fog servers and services cannot be static, and a dynamic approach is needed to adapt to changing workloads. Such issues are not specific to the Fog and have long been studied in the context of distributed and networked computer systems [7]–[12].

However, the servers and lightweight edge devices in the Fog call for simple dynamic algorithms without excessive decision making overhead. Thus we develop a fast decision algorithm for directing requests that originate at different devices towards multiple servers where services are located, without significant overhead for the edge devices and the servers, by exploiting the presence of Software Defined Network (SDN) controllers [13], [14] to provide a Service Management function in addition to packet routing.

The most advanced but costly approach is to migrate services between Fog nodes. This solution is worth using if there are very significant changes in network and server load, and in memory occupancy of the servers. Indeed service migration takes time and bandwidth; it limits the availability of services during migration and reduces the resources available to end users. In IoT networks in particular [3], due to the relatively steady nature of monitoring and actuating on cyberphysical infrastructures, significant changes in network usage are relatively infrequent. Therefore it may suffice to locate each service in a few replicate locations and optimally select its location for a given client's request. Different instances of a service can also be activated on-the-fly when replicas are installed at different system nodes. However, replicating services raises the issue of data consistency [15], and consistency control algorithms lead to overhead [16]. The optimization of such systems with respect to Quality of Service (QoS) and Energy Consumption using queueing theory was studied in [17]. The allocation of tasks to Cloud servers was considered in [18], [19] using Reinforcement Learning (RL) [20] and Deep Learning [21].

This paper discusses the allocation of users' requests for access to a given service s which is located at several $N(s) > 1$ different servers or Fog nodes. Our approach, which has been developed as part of a larger European project on smart and secure IoT network management [22], defines a relevant cost or Goal Function which includes the measured QoS, to make decisions in real time regarding the choice of the service's location, using reinforcement learning (RL) [20] to optimize the user's perceived QoS. We implement the algorithm as a Service Manager (SM) platform installed in a SDN controller, which is transparent to the end use. Its performance is illustrated with experiments which show its effectiveness in the presence of dynamic time-dependent changes in workload.

In the sequel, in Section II we first present a formalization of the optimization problem for the selection of an instance of a service for the request made by some user, when multiple instances of several services are located at the nodes of a Fog platform. Section III discusses the ML method that we use, and in Section III-B we detail the RL algorithm that is at the heart of the system that we have designed and tested. Section IV presents the specific example that we have experimentally tested in this paper, where the SM is in charge of selecting a particular location where a service request formulated by some user will be executed with the objective of optimizing the resulting QoS. The experimental results that we present show how the service requests are dynamically allocated, and re-allocated to another server if a given server becomes overloaded due to excessive workload. The final section is devoted to drawing some conclusions and suggesting directions for further work.

II. The Decision System

We consider a system consisting of N nodes $\{1, \ldots, N\}$ where the nodes can be connected via an underlying multi-hop Internet topology. Thus the N nodes can be viewed as an overlay network, or as Fog servers. Any two Fog nodes can communicate and transfer data and tasks to each other.

Services, such as data storage systems, named data servers, content providers or services that execute tasks, are located at these Fog nodes. Service requests are formulated by users, and can then be directed towards one of these nodes by the "Fog Manager" (FM) which is a decision system that may reside at each of the nodes, or which may itself reside at some other node. For the purposes of this paper, we do not dwell on where the FM resides and we viewed it as some form of transparent instantaneous decision system.

The general problem we formulate is about placing the set of users U and the set of services S at the various nodes or locations. Some user u at location $l(u) \in \{1, \ldots, N\}$ generates requests $R(u)$ to some service s so that $R(u) = s$. The location of s will be denoted $l(s) \in \{1, \ldots, N\}$. Generally users may be mobile, but will make a request from a specific location. On the other hand, a service s may be duplicated at a set of locations $L(s) \subset \{1, \ldots, N(s)\}$.

The request from u to $s \in L(R(u))$ is satisfied with after some transfer delay $T(l(u), l(R(u))$ which depends on the nodes where the user and service are located, and on the currently used network paths between them. Furthermore, the queueing plus service delay $D(.)$ needed to satisfy the request will also depend on the node $l(R(u)$ that services the request, and on its current load that we denote by $K(l(R(u)))$. Thus we will have some non-linear dependency $D(K(l(R(u)))$, to which we should add the load-dependent local delay at the node where u is connected, which we will denote $d(K(l(u)))$.

Therefore when a user u makes a request for a service $s = R(u)$ the purpose of the FM s to try to minimize an objective or Goal function of the form:

$$\begin{aligned} G(u, l(R(u))) &= T(l(u), l(R(u))) + D(K(l(R(u))) && (1)\\ &+ d(K(l(u))) + \alpha I(u, l(R(u))) && (2)\\ &+ \beta \mathbf{E}(u, R(u)), \end{aligned}$$

where $\alpha,\ \beta \geq 0$ are constants that weigh the relative importance of security and energy consumption within the overall cost, with respect to the other factors that concern the QoS. The security and energy consumption terms are defined as:

- $I(.)$ refers to a non-negative numerical value that characterizes the "insecurity" of having user u access service $R(u)$ at location $l(R(u))$. We note that this insecurity can actually be due to the user or its sensitivity, rather than the location, or it can be interpreted as depending on some risks or attacks that are related to the location $l(R(u))$, which is the more likely case.
- $E(.)$ refers to the resulting energy consumption, and:

$$\begin{aligned} \mathbf{E}(u, R(u)) &= E(l(u), l(R(u))) + E(K(l(R(u))) \\ &+ E(K(l(u))). \end{aligned}$$

The examples we provide in the sequel will be limited to QoS related optimization, so that we will not dwell on the values of $\alpha,\ \beta$ in the sequel.

The minimization of $G(u, R(u))$ will be carried out over all possible locations $l(R(u)) \in L(R(u))$. When we are free to instantiate the service $s = R(u)$ on any of the servers of the system, then we will obviously have $L(R(u)) = \{1, \ldots, N\}$.

The minimization of $G(u, R(u))$ is the optimization problem that is discussed in this paper, and we would tend to allocate the request $s = R(u)$ of user u to the node:

$$i^*(u, s) = \arg\min\{G(u, i) :\ s.t.\ i \in L(R(u))\}\ . \quad (3)$$

More restricted cases of this problem have been considered in earlier work. In [19], the services are duplicated at all the nodes, and requests emanate from a single node and are then dispatched to any one of the nodes using a recurrent Random Neural Network (RNN) [23] based RL scheme. Other work [18] uses a RNN based algorithm that considers both remote and local nodes so that the transfer of requests to remote nodes incurs a communication delay plus a processing delay, while local nodes have a congestion based queueing delay plus a request processing time. Note that in (2) each of the terms $D(K(l(R(u)))$ and $d(K(l(u)))$ can include both a queueing delay waiting for service at the node, and a service time.

III. Random Neural Network and Reinforcement Learning

Because the parameters in the Goal function can only be learned or estimated through measurement over some period of time, we propose a machine learning approach, and we first introduce the neural network model that will be used, which is recurrent, i.e. it contains feedback between its nodes. In fact, its adjacency graph is a fully connected directed graph on identical to the topology of the possible IP connections between nodes in the real system.

We use the RNN [24] because of its two important mathematical properties: it has a convenient closed form analytical solution in "product form", and it has an unique numerical solution despite its recurrent non-linear structure. Thus for a given set of input parameters it is guaranteed to provide a unique state and output value. We will associate one distinct neuron of the RNN for each of the N distinct nodes or servers where services may be placed, and the RNN will be used to compute the node to which a service request is directed.

An N neuron RNN is a probabilistic dynamical system whose state is represented by the vector of non-negative integers $K(t) = (K_1(t), \ldots, K_N(t))$ at time $t \geq 0$, where $K(t)$ is a vector random process. A particular value taken by $K(t)$ is denoted by the deterministic vector $k = (k_1, \ldots k_N)$. $K_i(t)$

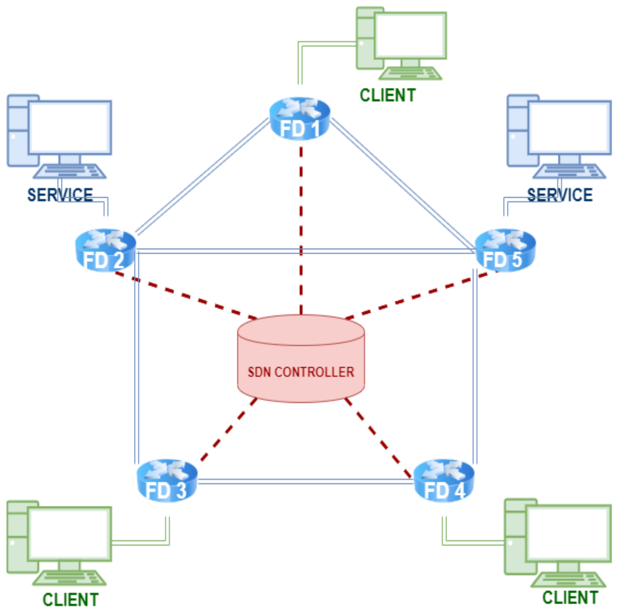

Fig. 1. The topology of a 5-node packet network with 6 inter-node links, and 5 attached servers, two of which support services and the three others support end users. The SDN controller communicates with every node and acts not just to establish network paths, but also to decide which service location or instance will be used by the service requests. This system is used as a test-bed for the experiments reported in this paper.

represents the "voltage" or potential of neuron i. The neurons are interconnected via excitatory and inhibitory weights that are denoted by $W_{ij}^{+} \geq 0$, $W_{ij}^{-} \geq 0$, respectively. These weights can be viewed as rates of spiking from any neuron i to any neuron j.

Each excitatory spike sent from i and arriving at time t to j will increase the value of $K_j(t)$ by $+1$, i.e. its effect will be $K_j(t^+) = K_j(t) + 1$. Similarly, each inhibitory spike sent from i to j at time t will have the following effect: $K_j(t^+) = max[K_j(t) - 1, 0]$. However a neuron i cn only send out spikes if its potential is positive, i.e. when $K_i(t) > 0$. Furthermore, when neuron i sends a spike to neuron j, then its own potential drops by 1, i.e. $K_i(t^+) = k_i(t) - 1$.

The key theorem concerning the RNN [23] states that:

$$\lim_{t\to\infty} Prob[K(t) = k] = \prod_{i=1}^{N} q_i^{k_i}(1 - q_i),\ where \quad (4)$$

$$q_i = \frac{\Lambda_i + \sum_{j=1}^{N} q_j W_{ji}^{+}}{\lambda_i + \sum_{j=1}^{N}[W_{ij}^{+} + W_{ij}^{-}] + \sum_{j=1}^{N} q_j W_{ji}^{-}}. \quad (5)$$

We also use r_i to denote the quantity $r_i = \sum_{j=1}^{N}[W_{ij}^{+} + W_{ij}^{-}]$, and we call it the "total firing rate" of neuron i.

Note that each decision is user and service dependent, and different users may have different locations in the network. Therefore in general we may have a distinct RNN for each user and service, and we can write:

$$q_i(u,s) = \frac{\Lambda_i(u,s) + \sum_{j=1}^{N} q_j(u,s) W_{ji}^{+}(u,s)}{\lambda_i(u,s) + r_i(u,s) + \sum_{j=1}^{N} q_j W_{ji}^{-}(u,s)}, \quad (6)$$

where $r_i(u,s) = \sum_{j=1}^{N}[W_{ij}^{+}(u,s) + W_{ij}^{-}(u,s)]$, is the "total firing rate" of neuron i.

Let $i^*(u,s) = \arg\max i\{q_i(u,s)\}$: we will consider that $i^*(u,s)$ is the node that is preferred by the decision algorithm to select the location of the service $s = R(u)$ requested by user u; hence it is in some sense the node that is estimated to provide the best performance to the current service request from user u for the service $s = R(u)$.

A. *Initialisation of the Recurrent RNN*

Before any data has been gathered, and before they are updated using the RL algorithm that we describe in the following section, the RNN weights should be set in a manner that makes all the $q_i(u,s) = 0.5$ to represent a situation where all possible choices are equally likely, and all weights are identical, i.e.

$$\begin{aligned} w &= W_{ij}^{+}(u,s) = W_{ij}^{-}(u,s), \quad (7)\\ \lambda &= \Lambda_i(u,s) = \lambda_i(u,s),\ \forall\ i,\ j,\ u,\ s, \end{aligned}$$

which will yield the equation:

$$0.5 = \frac{\lambda + 0.5Nw}{\lambda + 2.5Nw},\ or\ \lambda = 1.5Nw. \quad (8)$$

Thus to obtain $q_i(u,s) = 0.5$, we can set w to any value, as long as we also set $\lambda = 1.5Nw$.

B. *The Reinforcement Learning Algorithm*

The Goal function G or G' of (2) or (19) will be used with the RNN and a Reinforcement Learning (RL) algorithm to optimize the system. The objective is to choose the best node i where the service $s = R(u)$ requested by user u should be instantiated or located. We first define the *Reward* $R(u,s) = G(u,s)^{-1}$ or $R(u,s) = G'(u,s)^{-1}$ which must be maximized when the Goal is minimized. Successive values of $R(u,s)$ are measured, or measured and estimated. For instance, transfer times between the location of u and the different nodes in the network can be measured, and they do not depend on actually executing a user request for a service. Similarly, the execution time of a service at different locations for other users u', other than the actual user u, can be used to estimate $D(K(l(R(u)))$, while $d(K(l(u)))$ can be estimated by measuring the performance related to the local node where u is residing.

Successive values of the "reward" $R_l(u,s) = G_l(u,s)$, $l = 1, 2,\ ...$ will be obtained from the successive measured Goal values $G_l(u,s)$, $l = 1,\ 2,\ ...$ that are brought back by SPs and are used them compute "historical value" of the reward:

$$T_l(u,s) = \delta * T_{l-1}(u,s) + (1-\delta) * R_l(u,s), 0 < \delta < 1, \quad (9)$$

where $0 < \delta < 1$ is a responsiveness parameter that determines the importance of past historical values. Setting it to a high value will prevent the RNN from taking hasty decisions. The RNN weights are then updated as follows.

First save the current values of the sum of the weights $r_i(u,s) = \sum_{j=1}^{N}[W_{ij}^{+}(u,s) + W_{ij}^{-}(u,s)]$. Let k be the most recent selected "best" choice of the location for service s with regard to user u, i.e. $k = i^*(u,s)$ or $k = I^*(u,s)$. Then:

$$If \quad R_l(u,s) >= T_{l-1}(u,s) \quad then \quad for \quad j \neq k: \tag{10}$$

$$\begin{aligned} \forall i \neq k: \qquad & W^+_{ik}(u,s) \leftarrow W^+_{ik}(u,s) + R_l(u,s), \\ & W^-_{ij}(u,s) \leftarrow W^-_{ij}(u,s) + R_l(u,s), \end{aligned} \tag{11}$$

$$If \quad R_l(u,s) < T_{l-1}(u,s) \quad then \quad for \quad j \neq k: \tag{12}$$

$$\begin{aligned} \forall i \neq k: \qquad & W^-_{ik}(u,s) \leftarrow W^-_{ik}(u,s) + R_l(u,s), \\ & W^+_{ij}(u,s) \leftarrow W^+_{ij}(u,s) + R_l(u,s). \end{aligned} \tag{13}$$

After these updates, a normalization is carried out for all the weights, preventing them from constantly increasing:

$$W^+_{ij}(u,s) \leftarrow W^+_{ij}(u,s) \frac{r_i(u,s)}{\sum_{j=1}^N [W^+_{ij}(u,s) + W^-_{ij}(u,s)]}, \tag{14}$$

$$W^-_{ij}(u,s) \leftarrow W^-_{ij}(u,s) \frac{r_i(u,s)}{\sum_{j=1}^N [W^+_{ij}(u,s) + W^-_{ij}(u,s)]}. \tag{15}$$

Now with these updated values of the weights, we compute all the $q_i(u,s)$ using the system of equations (6), and obtain the new value of the "best location":

$$i^*(u,s) = \arg\max\{q_i(u,s)\}. \tag{16}$$

IV. Service Duplication at Several Locations

In our current implementation and experiments, we use a simpler Goal Function (2), where we aggregate the network transfer time and service delay into a single term:

$$Q(u,l(R(u))) = T(l(u),l(R(u))) + D(u,l(R(u))), \tag{17}$$

because these two quantities are measured in our experiments as one single value, which use as the Goal for the RL algorithm:

$$\begin{aligned} G(u,l(R(u))) \quad = \quad & Q(u,l(R(u)) + \alpha I(u,l(R(u)) \\ & + \beta \mathbf{E}(u,l(R(u))). \end{aligned} \tag{18}$$

The experimental platform on which these ideas have been implemented and tested is represented in Figure 1 where the five network nodes can be used to support either users or services. In this case we see that three nodes support users, while two nodes support services, and the six links that exist between nodes are also explicitly shown. Both the services and the users are in fact on separate machines which are connected to the network nodes.

A. Network Level Path Control

The system, both for network routing and for accessing services by specific users, is run by a SDN controller [2], [25] via a switch which is connected to each of the five network nodes as shown in Figure 1. The SDN controller uses OpenFlow Version 1.2-1.5 [26]. The SDN system in our test-bed was extended using the "cognitive packet routing algorithm" [27] to conduct smart measurements of network delays using "smart packets" (SP) so as to find network paths that minimize packet delays, similar to the approach in [28]. The SDN controller checks the network state each 5 seconds, and network paths can be changed at those times if significantly better paths are found that improve previously measured source-to-destination delay by over 30%.

B. Service Management

The SDN controller in our system is also in charge of the allocation of a user u's requests $R(u)$ to the locations $l(R(u))$. where the requested service is resident and may be satisfied. Within the SDN controller, for each user-service pair (u,s), we install a RNN which has a number of neurons identical to the number of locations where the service can be found, which we denote $N(s)$. For instance, in Figure 1 we have $N(s) = 2$.

The weights of the RNN for the pair (u,s) are updated using Reinforcement Learning as described in Section III-B, based on measurements sent to the SDN controller by each user, and specifically the user's own perceived average total response time, from the instant when the request $R(u)$ is sent by u to the location $l(R(u))$, to the instant when the successful response was received by the user u, which corresponds to the quantity $Q(u,l(R(u)))$ previously defined. Thus these experiments are based on learning using:

$$G'(u,l(R(u))) = Q(u,l(R(u))), \tag{19}$$

without using either the "insecurity factor" or the energy consumption. The RNN weights are updated according to the algorithm in Section III-B, where the choice of the optimum location from the values $q_i(u,s)$ for $1 \leq i \leq N(s)$.

From the user point of view this solution is completely transparent. The user u is given a configuration file which includes an IP address and the port $(IP, Port)$ on which the service s can be found. Note that this is an IP address which is unavailable at the network level. Each time u wants to connect to s, it connects to $(IP, Port)$. On the edge node where u is connected, the SDN controller changes $(IP, Port)$ to the IP address of the real location $l(s)$ of s. When the service ends and the resulting reply goes back from the location of the service to the user, the real IP address is changed back to the original "dummy' IP address provided in the configuration file.

C. Experimental Results

Our experiments show the ability of the system we have designed, to provide rapid adaptation to changes in the measured QoS at the nodes, is varied by turning on or off an additional program that overloads the processors at each server where the service is located. We have conducted numerous experiments on the test-bed of Figure 1, with and without the SM being turned on, where the user requests are generated by to the server in node $FD1$ (at the top of the figure) and generate a steady sequence of successive requests for service at a rate of 10 requests per second. The services are processed by the servers attached to nodes $FD2$ and $FD5$, and service requests have an approximate response time of 100 milliseconds when a server only deals with the service request without any additional load.

We first show the resulting measured response times, the SM is turned off in the upper curve (in orange) of Figure 2: with a sudden increase in workload due to an additional internal load on the server attached to node $FD5$ of Figure 1. When the SM is in use, the user will experience a sudden increase in its total response when the workload at the server is increased. In the lower curve of the same figure, another experiment shows the results when the SM is constantly turned: when the load at node $FD5$ suddenly increases, the response time for the user requests first increases, but after a transient of approximately $2,000$ ms, the total average response perceived by the user drops back to normal, because the SM changes the IP address that the user accesses to node $FD2$.

Figure 3 shows an experiment where the response time to service requests is measured at the user end is plotted against elapsed time for a large number of successive service requests. The SM is turned on throughout the experiment, and initially the service requests are being assigned to the server attached to node $FD5$. At roughly $40,000$ ms after the start of the experiment, the users' response time rises steeply because an additional external load is imposed on the server attached to node $FD5$, and then drops because the SM has transferred the user's requests from the server at node $FD5$ to the server at node $FD2$. At roughly $100,000$ ms, the overload at the server at node $FD5$ is turned off, and a similar overload is turned on at the server at node $FD2$: again we observe a high increase in measured response time and then a drop to "normal" because the SM has transferred the request to the server attached to node $FD5$. A similar switch occurs in the opposite direction at roughly $270,000$ ms showing that the SM reacts appropriately. Indeed, the SM that uses the algorithm we have described, effectively preserves the end users' QoS, in the presence of sudden changes in the additional load at the servers. In Figure 4 we show the result of an experiment where we attach a Raspberry Pi3B+ to node FD2, and an Intel NUC with i7 processor and 16 GB ram with Ubuntu 18.04 to the FD5 node, the latter being roughly four times faster than the Raspberry. The workload of each user is a program which computes the prime factors of a large random integer, and each job has a distinct compute times. The upper orange curve shows the average response time when the SM is disabled. The response time when the SM is enabled is shown on the blue curve, where the SM dynamically allocates the workload to the most lightly loaded server, resulting in lower response time after a transitory period.

V. Conclusions

We have presented a novel control algorithm and its implementation as a "Service Manager" (SM) that dynamically allocates service requests from end users to the location that can satisfy the service and minimize the overall average response time, using RL with a RNN. The system can also to minimize an objective or Goal Function, that includes Quality of Service, Energy Consumption and Security. An experimental test-bed based on a SDN controller that implements the SM has been used to test the resulting system's performance. Experiments

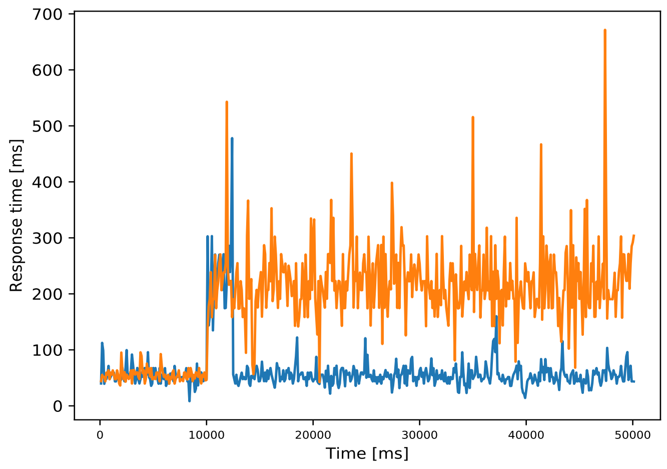

Fig. 2. The upper orange curve shows the increase in response times perceived by the end user when the SM is disabled, and the additional workload at the server attached to node $FD5$ is turned on. The blue curve shows the same experiment when the SM is enabled: after a brief transitory period, the user's measured response time drops to "normal" because the SM redirects the user requests to the server attached to node $FD2$.

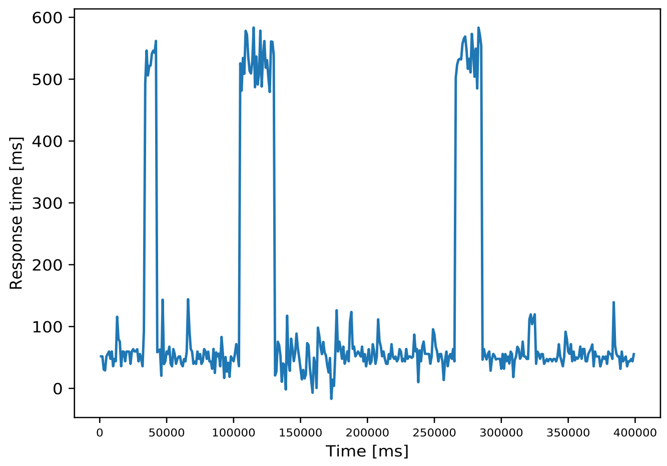

Fig. 3. We show that $40,000$ ms after the start of the experiment, the users' response time rises due to an additional external load on the server at node $FD5$, and later drops because the SM senses the overlaod and automatically transfers the users' requests to node $FD2$. At roughly $100,000$ ms, the overload at node $FD5$ is turned off, and the overload is turned on at node $FD2$, creating a sharp increase in response time and then a drop when the SM automatically transfers the requests to the server at node $FD5$. The experiment is repeated at roughly $270,000$ ms.

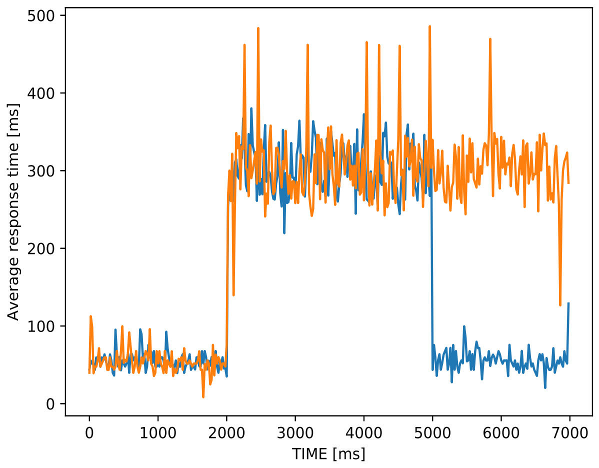

Fig. 4. This figure shows two experiments over the same time scale with a "small server" at $FD2$ and a more powerful server at $FD5$ of Figure 1. The measurements show response times to service requests with the SM disabled and service requests addressed at random with equal probability to the servers at $FD2$ and $FD5$. The measurement in orange show the response times when the SM is enabled, and the SM automatically allocates the service requests to the server that is more lightly loaded.

have illustrated the ability of our system to adapt in real time to the incoming load generated by the users, both with medium and high loads.

Currently, the IoT appears to be the main potential user of Fog services, however our proposed approach may also be used to support Base Stations for mobile users' video or other needs. For large systems we expect that the SDN router will avoid being congested by proactively distributing its advice at times when it is not re-routing the usual traffic rather than respond individually to each request. Future work will also address the effect of energy consumption and security, investigate the system's ability to adapt in the presence of competing end-users and multiple services, and develop a more general approach for handling multiple services.

ACKNOWLEDGEMENT

This work was supported by EU H2020 Program for the SerIoT Research & Innovation Project under G.A. No. 780139.

REFERENCES

[1] R. Buyya et al., "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Comput. Surv.*, vol. 51, no. 5, pp. 105:1–105:38, 2019. [Online]. Available: https://doi.org/10.1145/3241737

[2] A. Levin, K. Barabash, S. G. Y. Ben-Itzhak, and L. Schour, "Networking architecture for seamless cloud interoperability," in *2015 IEEE 8th International Conference on Cloud Computing, New York, NY*, 2015, pp. 1021–1024.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, August 2012, p. 13–16.

[4] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Profit-aware application placement for integrated fog-cloud computing environments," *J. Parallel Distrib. Comput.*, vol. 135, pp. 177–190, 2020. [Online]. Available: https://doi.org/10.1016/j.jpdc.2019.10.001

[5] C. Radoslav, "Cloud computing statistics 2019." [Online]. Available: https://techjury.net/stats-about/cloud-computing/#gref

[6] L. Goasduff, "Gartner says 5.8 billion enterprise and automotive iot endpoints will be in use in 2020," *Gartner Report*.

[7] C. Kim and H. Kameda, "An algorithm for optimal static load balancing in distributed computer systems," *IEEE Trans. Computers*, vol. 41, p. 381–384, 1992.

[8] H. Topcuouglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distributed Systems*, vol. 13, no. 3, p. 260–274, 2002.

[9] X. Zhu, X. Qin, and M. Qiu, "Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters," *IEEE Trans. Computers*, vol. 60, no. 6, p. 800–812, 2011.

[10] W. Tian, Y. Zhao, Y. Zhong, M. Xu, and C. Jing, "A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters," pp. 311–315, 2011.

[11] Z. Zhang and X. Zhang, "A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation," in *Proc. 2nd Int. Conf. Industrial Mechatronics Automation*, vol. 2, 2010, p. 240–243.

[12] S. Dobson et al., "A survey of autonomic communications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 1, no. 2, pp. 223–259, 2006.

[13] S. Bera, S. Misra, and A. V. Vasilakos, *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.

[14] J. Mambretti, J. Chen, and F. Yeh, "Next generation clouds, the chameleon cloud testbed, and software defined networking (sdn), 2015 international conference on cloud computing research and innovation (icccri)," pp. 73–79, 2015.

[15] E. Gelenbe and G. Hebrail, "A probability model of uncertainty in data bases," in *1986 IEEE Second International Conference on Data Engineering*. IEEE, 1986, pp. 328–333.

[16] E. Gelenbe and K. C. Sevcik, "Analysis of update synchronization for multiple copy data bases," *IEEE Trans. Computers*, vol. 28, no. 10, pp. 737–747, 1979. [Online]. Available: https://doi.org/10.1109/TC.1979.1675241

[17] E. Gelenbe and R. Lent, "Energy-qos trade-offs in mobile service selection," *Future Internet*, vol. 5, no. 2, pp. 128–139, 2013. [Online]. Available: https://doi.org/10.3390/fi5020128

[18] L. Wang, O. Brun, and E. Gelenbe, "Adaptive workload distribution for local and remote clouds," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2016, pp. 3984–3988.

[19] L. Wang and E. Gelenbe, "Adaptive dispatching of tasks in the cloud," *IEEE Trans. Cloud Computing*, vol. 6, no. 1, pp. 33–45, 2018. [Online]. Available: https://doi.org/10.1109/TCC.2015.2474406

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2nd Ed., 2018.

[21] Y. Yin, "Deep learning with the random neural network and its applications," *CoRR*, vol. abs/1810.08653, 2018. [Online]. Available: http://arxiv.org/abs/1810.08653

[22] J. Domanska et al., "Research and innovation action for the security of the internet of things: The seriot project," in *Recent Cybersecurity Research in Europe: Proceedings of the 2018 ISCIS Security Workshop, Imperial College London. Lecture Notes CCIS No. 821, Springer Verlag*, vol. 821, 2018.

[23] E. Gelenbe, "Random neural networks with negative and positive signals and product form solution," *Neural Computation*, vol. 1, no. 4, pp. 502–510, 1989.

[24] ——, "Learning in the recurrent random neural network," *Neural Computation*, vol. 5, pp. 154–164, 1993.

[25] S. Basterrech and G. Rubino, "A tutorial about random neural networks in supervised learning," in *Neural Network World*, vol. 25, no. 5, 2016, pp. 457–499.

[26] "Home page of onosproject - open source sdn controller." [Online]. Available: https://onosproject.org

[27] E. Gelenbe, "Steps toward self-aware networks," *Communications of the ACM*, vol. 52, no. 7, pp. 66–75, 2009.

[28] F. François and E. Gelenbe, "Towards a cognitive routing engine for software defined networks," in *2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, May 22-27*, 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/ICC.2016.7511138