# Enabling Service-Centric Networks for Cloudlets Using SDN

Ahmet Cihat Baktir
NETLAB
Department of Computer Engineering
Bogazici University
Istanbul, Turkey
Email: cihat.baktir@boun.edu.tr

Atay Ozgovde
Department of Computer Engineering
Galatasaray University
Istanbul, Turkey
Email: aozgovde@gsu.edu.tr

Cem Ersoy
NETLAB
Department of Computer Engineering
Bogazici University
Istanbul, Turkey
Email: ersoy@boun.edu.tr

***Abstract*—The current developments in smart devices, wearable gadgets and IoT (Internet-of-Things) are triggering a variety of novel use cases and services. Once the technology matures, a wide range of services is expected to be provided at the edge of the network in coordination with the cloud computing infrastructure. These services will be highly dynamic meaning that they can be served from edge computing facilities and from central cloud servers being transferred back and forth. Also, considering the mobility of the users and the varying demand, those services might need to be live migrated between nearby edge servers on-the-fly. This setup creates an environment that needs to be transparent to the end users. The current legacy networking paradigm, however, allow services to be reached by IP and port addresses, not by their content. Alternatively, in the service-centric approach the services themselves are handled independent of their location and the focus shifts to "what" instead of "where". Due to the complexities involved, it is not a straightforward task to establish service-centricity at the edge scenarios. As a remedy, this paper proposes a service-centric approach at the edge servers using orchestration capabilities offered by the Software-Defined Networking (SDN) technology. To demonstrate how SDN can help to alleviate the problem, an emulation environment is used in which northbound applications are implemented in order to setup the service-centric structure. The effect of service-centric approach is shown with the load balancing experiments where the performance of the proposed system is evaluated for various use cases.**

***Index Terms*—Software-defined networking, service-centric networks, information-centric networks, edge computing, cloudlet**

## I. Introduction

The emergence of specialized gadgets such as smart glasses, smart watches and interconnected home appliances changes the landscape dramatically not only from the end-user perspective but also from the technical characteristics of the way network processes information. An estimation states that 97 million wearable devices generated 15 petabytes of traffic per month in 2015 [1]. As a result of this trend, a novel computation infrastructure called edge computing with mobile users and geographically distributed resources is emerging through which novel use cases become feasible.

The promising solution of edge computing consists of deploying computational resources closer to the end users which lowers the service access latency. Low latency is critical for the smooth user experience for most of the envisioned edge computing scenarios. There are different proposals for bringing the computational power closer to the edge such as Fog Computing [2], cloudlets [3] and Mobile-Edge Computing [4]. All these proposals are based on similar principles and have the same objectives in general but differ according to the way they are streamlined for certain usage scenarios. Many application areas for these technologies have been discussed in the literature such as augmented reality, connected vehicles, body area networks (BAN), smart grid, and solving the network problems such as congestion [5]–[9]. The Mobile-Edge Computing is maintained by the cellular network operators and the target is their subscribed users. To maintain clarity throughout the text, cloudlets are chosen as the reference edge computing technology to present ideas and implementations in this work since it is an older proposal than Fog Computing [3]. It is worth noting that the notions developed can equally be well applied to the other edge computing approaches.

As the edge computing becomes more and more integrated into our daily lives, the number of services available via cloudlets will dramatically increase. In that case, not only the sheer volume but also the dynamicity introduced by the mobility of the users and variability of the demand need to be tackled. Each edge server node has only a limited computation power even when compared with a small-size enterprise datacenter. Due to the dynamicity and multiplicity of the service requests, the composition of the services available from the cloudlets will also be dynamic. In this context, a service request generated by a mobile user may not necessarily reside on the closest cloudlet and can in fact be located in a multitude of locations. For instance, the requested service may not be readily available at any cloudlet nearby and can in fact be started to be served from a distant cloud while being transferred to the cloudlet to be further served from there. Similarly, due to the mobility of the user, the service can be handed-off to a neighboring cloudlet while being executed using live migration techniques. Under these highly dynamic conditions imposed over edge services, it becomes a necessity to devise novel methods to discover and instantaneously track the services offered by edge servers while they are consumed by end users. However, the current

978-3-901882-89-0 @2017 IFIP

legacy networking paradigm becomes infeasible to achieve this aim as its design incorporates strict dependence of the services to their network locations. In contrast, our service-centric approach gets its motivation from this vision where the mobile and intermittent behavior of the services themselves becomes transparent to the end-user who simply wants to have access to the computational capacity that is known to exist somewhere at the edge.

This study proposes a service-centric edge computing solution that is enabled by Software-Defined Networking (SDN). SDN decouples the data plane from the control plane and provides network programming capabilities via the northbound interface [10]. Our proposal employs the capabilities offered by SDN and its de-facto standard OpenFlow to orchestrate the edge services transparently to the end-user. A service being addressable by its identifier rather than its network location is becoming a necessity and it is envisioned that the future networking protocol stack will incorporate inherent support for the service-centric approach [11]. However, as of today no such support is present in the legacy TCP/IP stack. As a remedy, our method enables service-centricity without any modification to the existing protocol stack by exploiting the TCP port number and rarely used IP header field called DSCP.

As a further contribution, our work assumes the realistic view that the mobile end-users exploit the resources of the cloudlet via computation offloading that can occur at the method/function resolution. Here, the mobile device executes some part of the code natively, while for some other part, it sends RPC/RMI like requests to the cloudlet to get it executed [12]. We coin the term "sub-service" to refer such RPC-like requests and use it extensively throughout our work. The term sub-service is important as it models the actual interaction granularity between the end devices and the cloudlet, which in turn enables to obtain realistic performance results. In our work, we do not only incorporate "sub-service" as a service request modality but also make it completely service-centric.

The remainder of this paper is organized as follows. Section II presents the related works. Section III introduces how to manage edge servers with SDN for service-centric networks. Section IV provides the details about the proposed system implementation. Section V presents the experiment design and performance evaluation with a discussion. Section VI concludes the paper with future directions.

## II. Related Works

Cloudlets defined as a computer system with high capacity resources and accessible by mobile end-users in the geographical vicinity for the services provided [3]. Besides, they enable a caching mechanism for providing a faster access to the data for requests [13]. Fog Computing [2], which is introduced by Cisco, is a suitable technology for the real-time requirements as a natural result of geographically distributed resources at the edge. On the other hand, ETSI (European Telecommunications Standards Institute) and other telecommunication organizations collaborated together and developed a paradigm called Mobile-Edge Computing (MEC) [4]. The vision of MEC is well aligned with the previously mentioned cloudlets and defined as a server deployed at the edge for providing specific services for mobile users and dissolving the congestion problem at the backbone of the cellular networks [14].

A mode of interaction offered between cloudlets and end-user devices is the "code offloading". MAUI is an example that aims to achieve efficient energy consumption by using code portability to create a replica of smart phone application and deploy it on a remote infrastructure [12]. It can estimate costs of each method so that it identifies the methods that may be executed remotely.

The terms information-centric networking (ICN) [15] and content-centric networking (CCN) [16] become more popular in the literature for addressing the problem of traditional IP-address or location-centric model. Although they share the same objectives and architectural features, there are some differences between them such as name resolution, routing, caching and Named Data Objects (NDO) granularity [17]. On the other hand, service-centric networks (SCN) differs from ICN through supporting service requests in addition to the content requests. Service migration, emergence of novel services, client mobility and dynamic environment are not aligned well with the current TCP/IP protocol stack so that SCN is proposed to support effective service management [18]. It is discussed that content-centric schemes should be generalized towards a service-centric architecture because the Future Internet is envisioned as a supporting mechanism for services such as file storage/retrieval, audio/video streaming and location-based services [11].

There are solutions presented for supporting the ICN structure with the help of SDN concepts and Salsano et al. provide both a long term solution for ICN without considering the current limitations in SDN and a short term solution to experiment the combination of ICN and SDN [19]. Since the SDN has still missing points to fully support the ICN environment, the long term solutions discuss several choices for the packet format.

Named Data Networking (NDN) [20] is a popular ICN architecture based on CCN. A study proposed by van Adrichem et al. integrates SDN into NDN for leveraging an application-specific forwarding mechanism [21]. In order to distinguish the traffic generated by ICN from traditional IP traffic, this study introduces a layer to OpenFlow and implements a specific communication channel and controller module that cooperates with the existing OpenFlow communication channel.

MOCHA [22] is a multi-tier architecture that is composed of mobile devices, cloudlets and cloud servers in order to enable the efficient process of big data. The authors also focus on the task partitioning, fixed and greedy approaches, where the identical sub-tasks can be performed by one or more servers.

There are also solutions for combining the ICN and SDN by utilizing the existing features. One of them is proposed to integrate an ICN solution to the OpenFlow networks which maps the content name carried in IP options into a tag transported in TCP and UDP port fields that can be used as

a matching field by the OpenFlow [23]. A similar approach is presented with a proposal which is named ContentFlow that enables the use of OpenFlow within an ICN structure by achieving the content mapping and routing over the traditional IP architecture [24]. The objective is achieved by defining the content through the combination of destination/source ports and IPv4 addresses because OpenFlow provides a flow-based forwarding mechanism, not a content-based approach.

Serval is a proposal for SCN which aims to provide a solution for providing services with multiple servers to the mobile users [18]. Their motivation is that the traditional network structure does not fit to the dynamic service environment. In order to mitigate this challenge, they propose the Serval architecture with a Service Access Layer above the network layer. This new layer provides name-based routing, decoupling of control and data planes, and other service-level operations.

It is observed that the recent proposals for integrating ICN or CCN with SDN introduce a modification for enabling the primitive functionalities and flawless cooperation. On the other hand, there are some proposals that integrates both technologies by exploiting the existing features and resources in order to provide compatibility with the current infrastructure. The main objectives of the solution proposed by our study are not modifying the TCP/IP stack and using the available protocols. Moreover, only a small set of studies in the literature focus on decomposing a service into components. These studies generally assume that the components are identical, instead of dividing the whole service into dissimilar fragments and analyzing them separately. On the other hand, since OpenFlow provides a message system for retrieving the flow-related statistics from the underlying infrastructure, most of the relevant operations consider the network resources. However, the controller and northbound applications additionally need to be aware of the CPU utilizations of the servers. The load balancing operations for considering either network or computation resources and comparison of them may be essential for overcoming the resource allocation problem.

## III. Edge Service Management and SDN

Our work envisions interactions with the edge servers with a sub-service granularity while maintaining service-centricness. The edge services are considered as a composition of sub-services. The main reason for decomposing a service to its sub-procedures is to provide a fine granularity environment for the users since a gadget may execute some of the sub-services locally and offload the remaining part of the task to a remote computation resource [12].

To provide more insight, consider the scenario where a security staff with a wearable smart glass at an airport entrance gate is expected to recognize the authorized airport personnel and passengers by executing the face recognition procedures. While some part of the procedures can be run natively on the device itself, to enable real-time performance some necessary sub-services can be requested from a cloudlet or multitude of cloudlets deployed in the vicinity. A sample face recognition service being composed of many sub-services distributed over the end-user's gadget and some cloudlets is depicted in Figure 1. It is shown that a subset of procedures are deployed on the first cloudlet and another subset (there can be common or distinct sub-services) are deployed on the second cloudlet and all of the sub-services are provided by the last cloudlet. In most of the cases, it is expected that the sub-services can be served from multitude of locations in order to increase the general performance of the system. This sample is an illustration for presenting the insight of the sub-service resolution and providing an example case for the sub-service deployment over cloudlets.

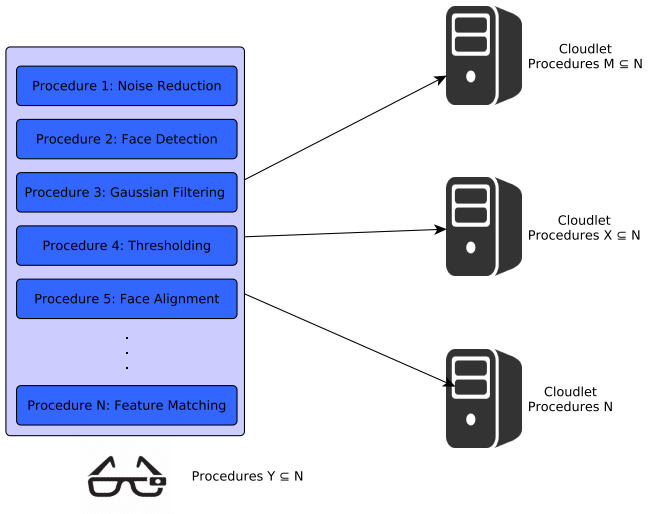



Fig. 1. Distribution of sub-services to the mobile device and cloudlets

Offering services at the level of sub-services allows the system designers to reflect the underlying code-offloading mechanisms into the networking level. Similar commercial attempts exist such as Amazon Lambda which allows users to deploy and execute a portion of a code on the Amazon computational resources [25].

The sub-services may be deployed over distinct cloudlets or multiple cloudlets may provide the same sub-service, so that each mobile user can request a sub-service from a less loaded cloudlet to further decrease the service delay.

In order to come up with an effective scheme that addresses both service-centricness and sub-service level messaging, SDN and OpenFlow capabilities are exploited. OpenFlow Extensible Match (OXM) [26] mechanism allows to use any of the 40 different fields for matching with the flow rules in the flow table of switches. The flexibility present in the OXM is not present in the legacy TCP/IP stack, however. As a remedy, to address sub-services in the TCP/IP domain before interacting with the SDN domain, DSCP (Differentiated services code point) field of the IP header is used. To address services to which the sub-services belong, TCP port numbers are used. Historically, 6-bit DSCP and 2-bit ECN (Explicit Congestion Notification) replaced the 8-bit Type of Service (ToS) field in order to

support QoS requirements and mitigate the congestion. ECN has an important role in the current network structure for congestion control but DSCP is still not being used commonly. The DSCP field is designed by IETF for the differentiated services. This behavior is per-hop and implemented locally in each router. When we consider the OpenFlow-enabled switches, they will not have such implementations in the near future since they are kept as simple as possible. Therefore, in this work the DSCP field is used with sub-service addressing to achieve the desired service management qualities for edge services.

The communication between the software-based controller and forwarding devices are provided through a protocol such as OpenFlow [26] which is capable of modifying the flow tables of the data plane devices. On the other hand, the controller communicates with customized applications through a northbound API and inter-operate to collaboratively define the network behavior.

## IV. Implementation of SDN-based Service-Centric Structure

The proposed system focuses on enabling the service-centric approach at the edge of the network. Therefore, the main objectives are achieving the communication between hosts using a sub-service resolution instead of the current IP address-based design and satisfying the QoS requirements of the end-users such as low service delay. The SDN as an enabling management layer can realize these objectives through a software-based controller and northbound applications where each one define a different feature for the whole system.

The general framework of the proposed system has the following components:

- Users and applications that demand different sub-services
- Data plane with OpenFlow-enabled network devices
- Cloudlets which are accessible through a LAN connection
- SDN controller
- Northbound applications for load balancing and service orchestration
- Global service locator

This section introduces the implementation of the system elements and how the framework operates when end-users generate interest packets for sub-services without having any prior knowledge about the location of the services, or IP addresses of the servers.

### A. System Operation

The implemented system works with sequential steps when a request is generated by the user at the edge. The main operations of the system are presented in Figure 2.

When the end-user generates a request for a sub-service in Step 1, by providing the destination port number and its DSCP value, the OpenFlow-enabled switch buffers this packet and forwards a replica of this packet to the controller in Step 2 if there is not any match within its flow table. The generated request needs to have a destination IP address but since the user application does not have any prior knowledge about the

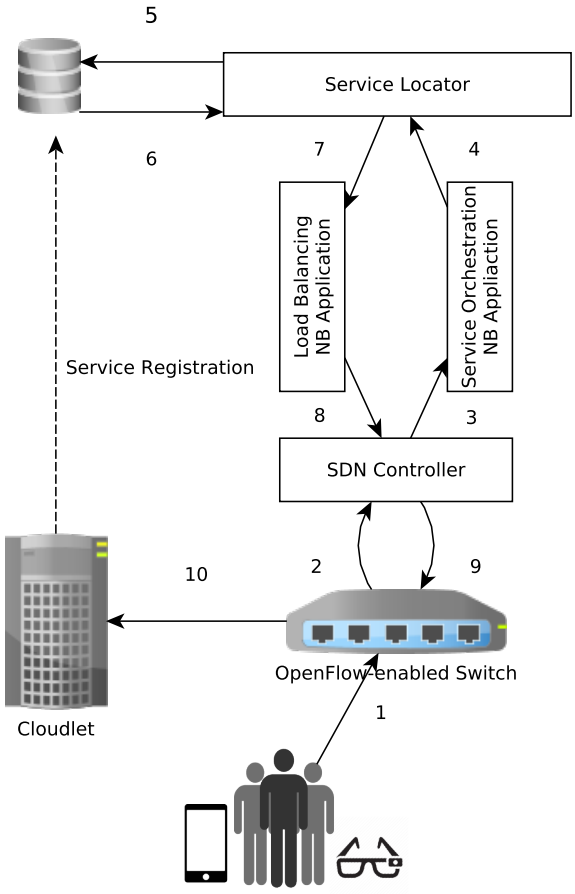



Fig. 2. The processes for forwarding the service request to the cloudlet

service location, it initially inserts a generic IP address defined by the system into the packet header which is then modified by the first switch within the network to be forwarded to the destined server.

The service orchestrator application receives this packet in Step 3 and interacts with the service locator in Step 4 to obtain the network location of the cloudlet that serves the sub-service in question - i.e. the sub-service indicated by the DSCP value. The locator queries its database with the specified destination port number and DSCP value and creates a list of IP addresses in Steps 5 and 6.

In general, the orchestrator takes the list of IP addresses and forwards it to the load balancer. These steps are merged as a single step in 7. Load balancer northbound application then internally decides on the destination server as explained in Section IV-C2 and sends that server's IP address to the controller in Step 8.

The controller installs a flow rule in Step 9 with a timeout value. In OpenFlow, there are two different timeout values implemented. First of them is called the "idle timeout" which states that if no flow arrives that matches a specific rule within

that timeout period, the flow rule is removed automatically. The other one is the "hard timeout" which causes to remove the flow rule after the timeout period, irrespective of the number of the matched flows. These parameters have a significant effect on the performance of the proposed solution.

The newly inserted flow rules indicate that any further request for a sub-service with the corresponding port number and DSCP value should be forwarded to the server by modifying the IP header field and inserting the destined cloudlet's IP address. After installing the flow rule, it sends an OpenFlow message (OFPT_PACKET_OUT) to the switch in order to forward the buffered request packet to the cloudlet after the same modifications in the header. Lastly, at Step 10, the incoming request is forwarded to cloudlet that provide the sub-service requested by the end-user. Besides, any cloudlet can make a sub-service registration at any time when it begins to provide that sub-service.

### B. The Service Locator

As illustrated in Figure 2, the service locator keeps track of the services, sub-services and their locations by introducing a mapping mechanism within its database. The database tuples are recorded through the service registration process initiated by cloudlets. When a cloudlet begins to provide a sub-service to the clients, it needs to inform the service locator to create a new record by specifying its own identity and the service information with corresponding field values. When there is an incoming request for a locating a sub-service, the locator queries the database with the given TCP port number and DSCP field, and gets the list of IP addresses that provide the requested sub-service.

### C. SDN Control Mechanism with Northbound Applications

OpenFlow-enabled switches (hardware or software-based) and cloudlets are off-the-shelf devices. The main contribution of our proposed solution is the customized control layer that is composed of the SDN controller and two different northbound applications. These applications operate collaboratively with each other and the controller for enabling the service-centricity and balancing the load among cloudlets.

The first northbound application is responsible for determining the possible destinations for an incoming sub-service request. Whenever an unmatched interest packet arrives to the switch, it is forwarded to the controller for a decision process. This northbound application operates in coordination with the service locator for determining the set of IP addresses that are available to execute the requested sub-service. In addition to this, it needs to be in communication with the other northbound application that is responsible for balancing the load among this set of cloudlets. The northbound applications require coherent interaction with each other for enabling the service-centricity and balancing the load for decreasing the service delay further.

*1) Service Orchestration:* When a sub-service request arrives to the OpenFlow-enabled switch with a specific destination port number and the DSCP field, the switch forwards any request that is unmatched with its flow rules to the controller. The controller handles this event by delegating it to the Service Orchestration northbound application. The Service Orchestration extracts the packet header fields, and generates a query to the service locator in order to find the set of cloudlets that provide the requested sub-service. After an internal query, the Service Locator provides the list of corresponding IP addresses to the Service Orchestrator.

Although the load on the cloudlets plays an important role in the performance of the framework, the load on the controller has also a significant impact. Therefore, tasks that require a considerable amount of processing are assigned to a northbound application instead of the controller itself. For this reason, the orchestrator helps locating the services and reducing the burden on the controller to prevent a possible performance bottleneck.

*2) Load Balancing:* The load balancing application operates in coordination with the service orchestrator. Once the service orchestrator receives the list of cloudlets that provide the requested sub-service from the locator, it assigns the task of deciding on the most feasible cloudlet to the load balancing northbound application. After the load balancing application decides on the server, it informs the SDN controller by sending the IP address of the cloudlet and the controller installs required flow rules.

There are four different methods implemented as distinct northbound applications for balancing the load among the cloudlets. The first one is by collecting the instantaneous CPU load from the cloudlets. Only the recent CPU loads are stored by the load balancer and it can find the least loaded one within an IP address set that is provided by the service locator. When the balancer receives the set of IP addresses, it assigns weights to these servers inversely proportional to their CPU loads. Therefore, if multiple requests arrive to the controller at the same instant, the requests are assigned to the servers according to these weights otherwise for each round a single cloudlet becomes heavily loaded.

The other method uses the already existing mechanisms in SDN where load balancing is done based on port statistics. The load balancing algorithm can collect statistics from the switches using OpenFlow messages and find the least loaded cloudlet in a specific instant by keeping the recent statistics.

The third load balancing application is implemented by following the Round Robin algorithm which is applied at the sub-service level. Apart from the three load balancing applications discussed, one last application used in the experiments assigns sub-service requests to cloudlets in random order.

## V. Performance Evaluation and Load Balancing

This section presents the experimental setup and the performance evaluation of the proposed infrastructure under various load.

The proposed system is implemented over the Mininet evaluation environment which is configured on a computer with Intel i5-3210M 2.5 GHz CPU and 4GB main memory. Ryu is chosen as the SDN controller because of its support

for the OpenFlow versions that includes OXM mechanism. The system is implemented in compliance with OpenFlow v1.3 specification. Within the Mininet environment, the most recent version of OpenvSwitch (v2.5) is used for the network forwarding devices at the data layer.

In the initial phase of the experiments, the service-centric behavior of the proposal is demonstrated. For the second part, experiments are conducted for evaluating the performance of different load balancing northbound applications in order to minimize the service delay. Each experiment is repeated for 5 times and an average of 100000 sub-service requests are included for each run.

### A. Enabling the Service-Centric Approach for the Cloudlets

The first objective of the framework is to enable the end-users requesting a sub-service without depending on the network location - i.e. the IP address. Therefore, the initial experiment setup is conducted in order to present this behavior of the implemented system.

TABLE I. The sub-service deployments for the initial experiment.

| Cloudlet | Service | Sub-service | IP Address |
|---|---|---|---|
| Cloudlet 1 | Port: 50006 | DSCP: 4 | 10.0.0.1 |
| Cloudlet 2 | Port: 50006 | DSCP: 8 | 10.0.0.2 |
| Cloudlet 3 | Port: 50006 | DSCP: 12 | 10.0.0.3 |

A single service experiment is designed where the service is defined with 50006 port number and its 3 sub-services are deployed over 3 different cloudlets. The sub-service deployments and scenario are summarized in Table I. According to the scenario, a single client that have access to the cloudlets requests all of these sub-services sequentially to accomplish the whole service.

The peaks shown in Figure 3 correspond to the exact times when the sub-service request arrives to the corresponding cloudlet so that the CPU load increases to execute the sub-service. The client application generates a request packet with the corresponding destination port number and DSCP value without any knowledge about the destination IP address. Therefore, it embeds a generic IP address defined by the system which is then modified by the OpenvSwitch on the path with the destined server's real IP address. As observed by analyzing the peaks in the graph, the sub-service requests are forwarded to the accurate cloudlets that satisfy the sub-service demand. In other words, the increase in the CPU load of a server is an indicator of a request for a sub-service has recently arrived and it is being completed at that time.

At the beginning, the cloudlets' CPU loads are close to 0, until the user application generates the initial request. According to the implementation, it is expected to execute the first task at Cloudlet 1 since it is only served by there. Between 15th and 30th seconds, the first sub-service request (DSCP 4) is forwarded to Cloudlet 1 by the SDN controller and it is executed within that period. Then the client application

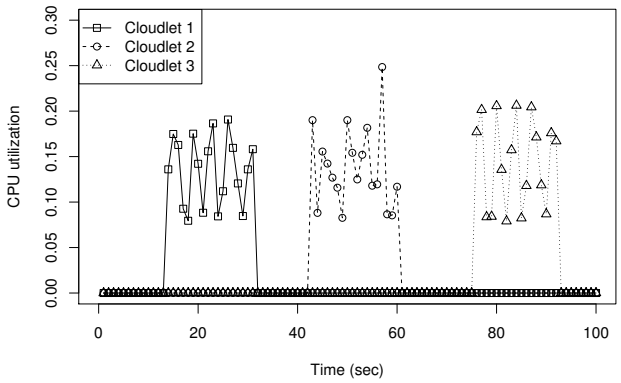



Fig. 3. Demonstration of the service-centric behavior at the corresponding cloudlets

generates a request for the second sub-service (DSCP 8) which is executed by Cloudlet 2 between 40th and 60th seconds. Lastly, the application generates the latest sub-service (DSCP 12) request which is provided by Cloudlet 3 and the request is forwarded to the corresponding server as observed.

As depicted, the system can resolve the locations of the sub-services that are requested by the user application and forward them accordingly. Thus, the service-centricness is enabled through the usage of port numbers and DSCP values.

### B. Minimizing Service Delay with Load Balancing

In order to enable load balancing among the cloudlets, a sub-service needs to be distributed over at least two servers. Doing so, a request can be forwarded to the cloudlet with some criteria depending on the type of northbound application for minimizing the service delay. It is worth noting that the delay for each sub-service is considered as the "service delay" since a client can request sub-services independent from each other.

In the experiment setup, varying number of mobile clients are assumed to request sub-services from five different cloudlets that are located in the vicinity. The network and computation resources for the cloudlets are identical to each other. As the computation delay is the dominant factor here, the network delay is negligible. The service that is identified with port number 50006 has 10 different sub-services with DSCP values 4, 8, 12, 16, 20, 24, 28, 32, 36 and 40. The distribution of these sub-services and the experiment setup are shown in Figure 4. There are 40 distinct request types defined with different combinations of these 10 sub-services as shown in Eq. 1 and Eq. 2 where *SS* stands for the set of sub-services and each $Request_i$ represents a request type consisting of a subset of *SS*. For evaluating the performance under different user behavior, each client requests services by randomly selecting one of these subsets. After all sub-services within that subset is completed, the client creates a $Request_i$ randomly again and request them sequentially. It is important to mention that each sub-service is unique and the amount

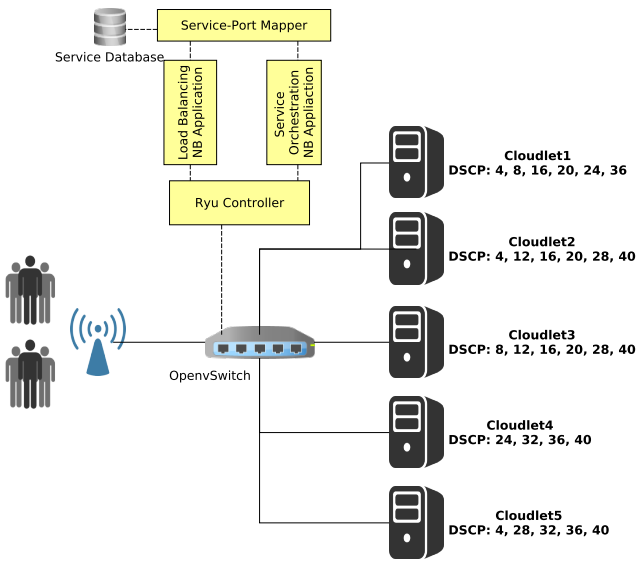



Fig. 4. The experiment setup for the performance evaluation of the framework

of generated computation load on the cloudlets is distinctive. None of them are IO-intensive sub-services so that the load on the CPU is only caused by the related computations.

$$SS = \{4, 8, 12, 16, 20, 24, 28, 32, 36, 40\} \quad (1)$$

$$Request_i \subseteq SS \text{ where } \forall i,\ 1 \leq i \leq 40 \quad (2)$$

As stated in Section IV, there are four different load balancing methods that are implemented and can be compared to analyze their effect on the service delays. The first of them is based on the CPU loads of the cloudlets, the second of them is Round Robin at sub-service resolution, the third of them is based on OpenFlow statistics request message

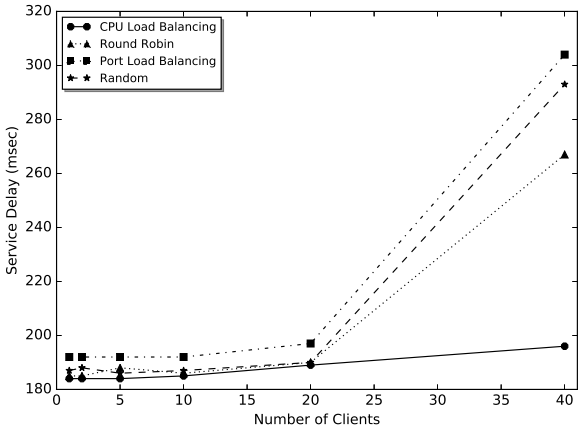
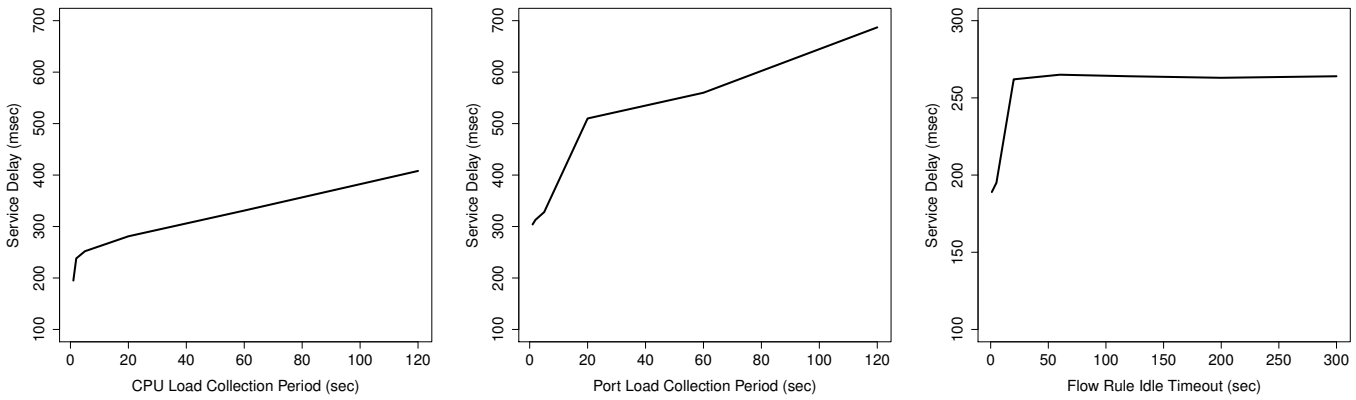



Fig. 5. The effect of load balancing applications with different number of clients on service delay

(OFP_PORT_MULTIPART_REQUEST) sent to the switches which reply back with the port statistics (OFP_PORT_STATS) including the number of packets or bytes sent over a port. The last one is assigning requests to the cloudlets randomly.

Figure 5 depicts the effect of user traffic on service delay. In this setting, flow rules have 5 second idle timeout and northbound application collects CPU or port information for every 1 second. For less than 20 clients, all of the methods provide approximately the same service delay while CPU load provides the least. However, as the number of clients increases, port load balancing, Round Robin and random assignment result in higher service delays where the CPU load balancing still provides the lowest service delay and it is not affected much with the increasing number of clients. It is observed that the methods except CPU load balancing shows the similar behavior with the various number of clients. Since each sub-service generates different amount of load on the cloudlets, the load balancing approach that considers the computational resource performs the best among all. It is able to track all the changes of cloudlet utilization, thus assigning the sub-service to the least-loaded cloudlet results in the lowest service delay. In other words, lower CPU usage leads to the instant execution of a sub-service request. On the other hand, balancing the load according to the port load collection results in the worst performance. The main reason for this situation is that the number of bytes or number of packets forwarded by a single port is identical for all of the sub-services. Thus, it does not reflect the main ongoing operations within the network. However, it performs even worse than the random sub-service assignment. Collecting the port statistics through OpenFlow messages generates an extra load on the switch and within the network but random assignment and Round Robin do not cause such an overhead. For achieving the most accurate and complete results, both network and computation resources can be considered together.

The frequency of the CPU load statistics collection from the servers and similarly the port load collection from the switches have an effect on the performance of these methods. To reveal this effect, the same environment is utilized with 40 clients where flow rules are installed with 5 seconds idle timeout. The effect of the time period between each load collection and balancing the load according to the recent values are shown in 6a and 6b. The minimum period of load collection is 1 second for both methods. Figure 6a shows that as the time period between two successive load collection increases considering the CPU cycles, the service delay increases because flow rules are installed according to the latest CPU load information of the cloudlets and as the period increases, it does not reflect the recent situation of the loads on the servers. The same information can be inferred for the port load collection in Figure 6b by sending and receiving OpenFlow messages. As the time period is increased for two consecutive OpenFlow statistics request messages, it tends to increase the service delay. Although the smaller period results in higher numbers of OpenFlow messages within the network that can cause congestion at specific points or increase the load on the




(a) The effect of CPU load collection period

(b) The effect of port load collection period

(c) Effect of flow rule idle timeout on service delay for CPU load balancing

Fig. 6. The effects of load collection periods and idle timeout on service delay

switches, it does not become the dominant factor that refers to a large service delay. Hence, it is extracted that the time period of load collection by both servers and switches have an important effect on the performance of the framework.

Another system parameter is related to the flow rules installed by the controller. Since the best performance is provided by the CPU load balancing, the effect of the idle timeout is observed on this method. The number of clients is fixed to 40 and the load collection period is configured as 1 second. The effect of changing the idle timeout for CPU load balancing is presented in Figure 6c. As the idle timeout increases from 1 (minimum idle timeout) to 30, it increases the service delay but increasing idle timeout further does not affect the service delay. The reason for initial increase in the service delay is caused by matching with the old flow rules installed by the controller. Therefore, as the flow rule idle timeout increases, the number of matching packets also increases. The system is highly dynamic and the loads on the cloudlets change instantly. Therefore, forwarding the requests according to an older flow rule does not reflect the recent condition of the cloudlets. After idle timeout reaches to 30 seconds, the service delay does not increase further because the average inter-arrival time between two consecutive requests is not higher than 30 seconds. Thus, the flow rules are never timed-out and the same rules are matched from the beginning. Actually this behavior heavily depends on the traffic characteristics and the user behavior. As the inter-arrival time changes, the point where the service delay becomes stable may change but the minimum service delay is achieved with the minimum idle timeout in every case. In general, it can be said that increasing the flow rule idle timeout also increases the service delay until a certain point then the service delay does not change with the increasing timeout value. As a result, this parameter is one of the key indicators of the system that affect the performance but it needs to be optimized according to the user behavior.

## VI. Conclusion

The emergence of new smart devices, wearable gadgets and popularity of IoT devices empower the novel use cases and scenarios which were not feasible before. For this reason, it is envisioned that the Future Internet will be service-centric with a focus on the service itself instead of the location. Since there are different QoS requirements and most of the services are not delay-tolerant, it is inevitable to bring the remote computation resources to the edge.

This study proposes a framework that utilizes SDN as an orchestrator of service-centric behavior at the edge of the network. The characteristics of SDN makes it one of the best candidates that facilitate service-centric structure for cloudlets and orchestrate the heterogeneous environment where services and users are geographically distributed.

Fully supporting the service-centric structure and providing compatibility between technologies require disruptive changes to the TCP/IP networking stack. Alternatively, this study proposes a solution that utilizes the current potential of SDN to leverage the service-centric approach for cloudlets.

The northbound applications and supportive mechanisms are implemented and experiments are conducted. Four different load balancing methodologies are implemented in order to satisfy the QoS requirements for latency-intolerant applications through decreasing the service delay as much as possible. The effects of the several system parameters on the performance are also analyzed.

This promising solution without modifying the existing protocol stack will pave the road for a more extensive framework in the future. However, it is expected that new protocols may appear in the near future for the variety of the mobile station.

## Acknowledgment

This work is supported by the Bogazici University Research under the Fund Grant No. 12663 and the Galatasaray University Research Foundation under the Grant No. 16.401.002.

## REFERENCES


[1] "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015-2020 White Paper, (2016, Februrary 3) [Online]. Available:," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html.

[2] 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, 2012, pp. 13–16.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[4] "European Telecommunications Standards Institute, Mobile-Edge Computing – Introductory Technical White Paper, [Online]. Available:," https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf.

[5] M. Quwaider and Y. Jararweh, "Cloudlet-based for big data collection in body area networks," in *Internet Technology and Secured Transactions (ICITST), 2013 8th International Conference for*. IEEE, 2013, pp. 137–141.

[6] V. S. Achanta, N. T. Sureshbabu, V. Thomas, M. L. Sahitya, and S. Rao, "Cloudlet-based multi-lingual dictionaries," in *Services in Emerging Markets (ICSEM), 2012 Third International Conference on*. IEEE, 2012, pp. 30–36.

[7] I. Stojmenovic and S. Wen, "The fog computing paradigm: Scenarios and security issues," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE, 2014, pp. 1–8.

[8] S. Yi, C. Li, and Q. Li, "A survey of fog computing: concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*. ACM, 2015, pp. 37–42.

[9] S. Nunna, A. Kousaridas, M. Ibrahim, M. Dillinger, C. Thuemmler, H. Feussner, and A. Schneider, "Enabling real-time context-aware collaboration through 5g and mobile edge computing," in *Information Technology-New Generations (ITNG), 2015 12th International Conference on*. IEEE, 2015, pp. 601–605.

[10] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[11] T. Braun, A. Mauthe, and V. Siris, "Service-centric networking extensions," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 583–590.

[12] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 49–62.

[13] U. Shaukat, E. Ahmed, Z. Anwar, and F. Xia, "Cloudlet deployment in local wireless networks: Motivation, architectures, applications, and open challenges," *Journal of Network and Computer Applications*, vol. 62, pp. 18–40, 2016.

[14] M. Satyanarayanan, R. Schuster, M. Ebling, G. Fettweis, H. Flinck, K. Joshi, and K. Sabnani, "An open ecosystem for mobile-cloud convergence," *IEEE Communications Magazine*, vol. 53, no. 3, pp. 63–70, 2015.

[15] M. Amadeo, C. Campolo, J. Quevedo, D. Corujo, A. Molinaro, A. Iera, R. L. Aguiar, and A. V. Vasilakos, "Information-centric networking for the internet of things: challenges and opportunities," *IEEE Network*, vol. 30, no. 2, pp. 92–100, 2016.

[16] S. Charpinel, C. A. S. Santos, A. B. Vieira, R. Villaca, and M. Martinello, "Sdccn: A novel software defined content-centric networking approach," in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2016, pp. 87–94.

[17] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.

[18] E. Nordström, D. Shue, P. Gopalan, R. Kiefer, M. Arye, S. Y. Ko, J. Rexford, and M. J. Freedman, "Serval: An end-host stack for service-centric networking," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 7–7.

[19] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri, "Information centric networking over SDN and openflow: Architectural aspects and experiments on the OFELIA testbed," *Computer Networks*, vol. 57, no. 16, pp. 3207–3221, 2013.

[20] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[21] N. L. van Adrichem and F. A. Kuipers, "Ndnflow: software-defined named data networking," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*. IEEE, 2015, pp. 1–5.

[22] T. Soyata, R. Muraleedharan, J. Langdon, C. Funai, S. Ames, M. Kwon, and W. Heinzelman, "Combat: mobile-cloud-based compute/communications infrastructure for battlefield applications," in *SPIE defense, security, and sensing*. International Society for Optics and Photonics, 2012, pp. 84 030K–84 030K.

[23] N. B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, "An openflow-based testbed for information centric networking," in *Future Network & Mobile Summit (FutureNetw), 2012*. IEEE, 2012, pp. 1–9.

[24] A. Chanda and C. Westphal, "Contentflow: Mapping content to flows in software defined networks," *arXiv preprint arXiv:1302.1493*, 2013.

[25] "Amazon, AWS Lambda, [Online]. Available:," https://aws.amazon.com/lambda/details/.

[26] "Open Networking Foundation, OpenFlow, [Online]. Available:," https://www.opennetworking.org/sdn-resources/openflow.