# Improving Quality of Service in Edge Computing Networks

Colin Rabe
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
rabex028@morris.umn.edu

## ABSTRACT

Edge computing is a data processing architecture in which a network of servers can store and process data between the cloud and devices that are connected to it. This type of network is useful because data does not need to travel as far to be processed as it would in conventional cloud computing, so results can be returned much faster. It is considered to be one of the enabling technologies for the expansion of the Internet of Things. This paper examines two techniques, called task offloading and service migration, that are being used in current models of edge computing. These techniques are designed to distribute work more efficiently within the network, which results in better quality of service for users.

## 1. INTRODUCTION

The pace at which computers are being connected to the Internet is rapidly accelerating. Estimates show that 50 billion devices will be connected to the Internet by the year 2020, with ten times that number in 2025 [3]. This trend indicates the increasing relevance of the *Internet of Things* (IoT) concept. IoT refers to objects capable of performing computation that can connect to each other to share information [3]. These objects include not only desktop computers and smart phones, but also vehicles, appliances, sensors, medical devices, and more. The ability for such devices to share information has the potential to make our lives significantly more convenient, but comes with a challenge. Huge quantities of data tend to be produced by these objects when they perform their services. Ni et. al note that "the large amounts of data result in heavy network congestion and complicated processing load on devices and control systems" [8].

Up until now, a common way to deal with large amounts of generated data has been to use cloud computing to process it. In cloud computing, devices can send their data to a powerful, but likely remote, computer that can process their data for them. However, for many new applications involving the IoT, cloud computing would not deliver results fast enough for devices because of the limited bandwidth of the connections and physical distance between the cloud and devices [6]. An alternative approach to address the problem is an emerging technology called edge computing. Edge computing is a paradigm in which computers that can process the data of IoT devices are located much closer geographically to these devices than the cloud is, i.e. at the edge of a network [6].

In this paper, we describe two different types of edge computing that exist, which are called fog computing and mobile edge computing (MEC). We will examine techniques respective to each type that are presented by researchers Yousefpour et. al [11] and Taleb et. al [7] for improving the efficiency of these networks and thus the quality of service to devices connected to them. In the next section, we will provide an overview of both fog and mobile edge computing, introduce relevant networking concepts and terminology, and briefly cover virtual machines. In Section 3, a technique for improving fog computing networks will be explained, and we will discuss the results of a simulation that was built for it. In Section 4, a technique for improving mobile edge computing networks will be explained, and we will again discuss the results of a simulation that was created to test the technique. Each of these techniques was effective in increasing the performance of the networks they were used in, and this increased performance corresponds with higher quality of service provided to users.

## 2. BACKGROUND

In the context of computer technology, a network refers to connected computers that can share information with each other [5]. Computing devices within a network are also known as nodes, while the connections between them are called links. One type of node that is particularly useful in networks is called a server. Servers help provide resources to other nodes within the network using resource sharing. With resource sharing, physical assets (such as more powerful hardware) and logical assets (data or software) can be accessed by any node in a network using servers [5]. Another important type of node is called a router. Routers serve the purpose of allowing separate networks to communicate with each other [5].

Data is sent between nodes in units which are called packets [5]. Nodes in a network cannot communicate with each other instantaneously; there are different types of delays that are involved when sending packets. Propagation delay, also called latency, for a link is the amount of time it takes for a packet to be sent from one node to another using a particular link [4]. Transmission delay is the amount of time it takes to propel all of the information in a packet from a node into a link.


*UMM CSci Senior Seminar Conference, November 2018* Morris, MN.

## 2.1 Edge Computing Networks

The edge of a network, as defined by Shi et. al in [6], can be any point along a link between computing devices and cloud servers that network resources, such as servers or routers, are found. Edge computing is then defined as "the enabling technologies allowing computation to be performed at the edge of the network..." [6].

The first type of edge computing is called fog computing. In fog computing, a network of nodes consists of three distinct layers of nodes arranged as a hierarchy. An IoT layer exists on the bottom, a fog layer in the middle, and a cloud layer on top [11]. The IoT layer is solely comprised of IoT devices that can communicate with the upper two layers. The fog layer contains fog nodes, which are typically routers (or other similar network connection devices) that contain embedded servers within them [1]. Fog nodes are located near IoT nodes and can process IoT requests that do not require permanent storage or would benefit from very quick responses. Finally, the cloud layer includes cloud servers, to which the IoT layer can send more complicated tasks. The IoT layer does not necessarily need to communicate through the fog layer to reach the cloud layer; requests can be sent to any layer directly.

The second type of edge computing is called mobile edge computing (MEC). MEC is similar to fog computing in that there is an intermediate layer of servers that exist between the cloud and IoT devices. However, MEC is designed specifically for devices that are mobile. In particular, MEC servers are designed to communicate with cellular network systems and are typically located near the towers for this infrastructure [1]. Another key difference between fog computing and mobile edge computing is that MEC servers make use of virtualization, whereas fog nodes do not.

## 2.2 Virtualization Layers

Virtual machines are simulations of computer processes, such as applications or operating systems, that have been installed and are running inside of another computer. They have the same behavior that an independent machine running those processes would have [10]. They are useful in MEC networks because they help compartmentalize huge amounts of data, which makes it easier to manage.

In the model of MEC that we studied, there are three different layers of virtualization that are used in the MEC nodes. The first layer is called the base, and consists of the operating system, or kernel, of the virtual machine. The next is called the application layer. It contains only data that is necessary to run an application. The last layer is known as the instance layer. It contains specific information about the state of an application. An example of a virtual machine that uses all three of these layers would be a Windows operating system with Microsoft Office installed on it running with multiple documents open. Respectively, these can represent the base, application, and instance layers of a virtual machine that can be run inside a MEC node.

# 3. FOG OFFLOADING

The first optimization technique for edge computing networks that is discussed is fog offloading. It is only applicable for fog computing networks. The goal of fog offloading is to reduce the service delay for IoT devices [11]. Service delay is the length of time it takes for a response to be received by an IoT device after it has made a request to a fog node. Often, these requests involve data analytics tasks across wide applications, for example, forest fire detection and air quality monitoring [3]. Fog offloading reduces service delay by approximating which fog node can complete a request the fastest, and then sending the request to that optimal node. If too many nodes are busy, or if the request is too complicated, the request is sent to the cloud instead of a fog node. A node that can complete a task the fastest depends on the complexity of the request, the number of tasks already waiting to be processed by a node, the computational power of a node, and the distance from a node to the device that made a request.

Fog nodes require time to process each request they receive, which is known as processing delay. In the model proposed by Yousefpour et. al, tasks can be categorized into light tasks and heavy tasks based on their average processing times [11]. For example, a light task might be a sensor needing to estimate the average temperature of a room (a simple calculation), while a heavy task could be a traffic camera requesting the licence plate of a car to be read from an image it has taken (image processing) [11]. A node can identify whether a request is light or heavy by checking the first packet in a request, which records its type. If a fog node receives multiple requests, they can wait in a queue inside of the node until they are ready to be processed. The estimated waiting time ($W$) for a node is the amount of time it will take for a node to process all of the requests in its queue, plus the amount of time it will take for it to complete the request it is currently working on [11]. $W$ can be estimated by keeping track of the average processing times of light and heavy tasks and counting the types of tasks in the queue. Each node $j$ has a threshold value $\theta_j$ that is checked every time a node receives a new request. If the fog node has a smaller estimated waiting time than this value, the fog node will begin working on the new request, or add it to its queue. If the node has a greater estimated waiting time than the threshold, the request will be offloaded to a neighbor of the fog node, or to the cloud. The number of times a request is offloaded is maintained as $N_{fwd}$. If $N_{fwd}$ exceeds a value $e_{\mathcal{M}}$ called the *offload limit*, then the request is sent to the cloud instead of a neighboring fog node. Here, $e$ stands for the offload limit, and $\mathcal{M}$ stands for a given fog network.

In order to determine which neighboring node to offload a task to, in the model created by Yousefpour et. al nodes communicate their estimated waiting times to each other, as well as the propagation delays between each pair. The neighboring node with the smallest sum of estimated waiting time plus propagation delay is chosen as the node to which the request is offloaded.

## 3.1 Service Delay

The model presented by Yousefpour et. al describes service delay with the equation below.

$$\begin{aligned} d_i = {} & p_i^I \cdot (A_i) + p_i^F \cdot (X_{ij}^{IF} + Y_{ij}^{IF} + L_{ij}) \\ & + p_i^C \cdot (X_{ik}^{IC} + Y_{ij}^{IC} + H_k + X_{ki}^{CI} + Y_{ki}^{CI}), \\ & j = f(i); \;\; k = g(i) \end{aligned} \tag{1}$$

Here, $d_i$ represents service delay for an IoT node $i$ working on a task. Then $p_i^I$ is the probability that an IoT node $i$ can complete its task without using the fog network or the cloud, and $A_i$ is the average amount of time it takes for the IoT

node $i$ to complete a task. $p_i^F$ is the probability that the IoT node will send the task to the fog layer. $X_{ij}^{IF}$ and $Y_{ij}^{IF}$ respectively represent the propagation and transmission delays for IoT node $i$ to send a request to fog node $j$. The transmission delay is found by finding the sum of each transmission delay across all of the links between IoT node $i$ and fog node $j$ (if there is more than one link). $L_{ij}$ is the amount of time it takes for the fog layer to process IoT node $i$'s request that is initially sent to fog node $j$. $L_{ij}$ is covered in greater detail in the next subsection.

The second line of this equation handles the case where an IoT node's request skips the fog layer and is immediately sent to the cloud to be processed. $p_i^C$ is the probability of of this event happening. $X_{ik}^{IC}$ and $Y_{ij}^{IC}$ are the propagation and transmission delays between the IoT node $i$ and cloud server $k$. Accordingly, $X_{ki}^{CI}$ and $Y_{ki}^{CI}$ are the propagation and transmission delays from the cloud server $k$ back to the IoT node $i$. As with the transmission delay in the fog layer, the transmission delays $Y_{ij}^{IC}$ and $Y_{ki}^{CI}$ are equivalent to the sums of the transmission delays for each link between the the cloud server $k$ and IoT node $i$.

$H_k$ is the average delay for the cloud server to process a task sent to it. The last components of the service delay equation are the functions $j = f(i)$ and $k = g(i)$. Given an IoT node $i$, they assign a particular fog node $j$ and cloud server $k$ to which it will send its requests. For example, these functions may be defined as assigning IoT node $i$ to the fog node and cloud server with the least amount of propagation delay among all of the choices [11].

## 3.2 Fog Layer Delay

In the equation below, $L_{ij}$ from the service delay equation is defined as:

$$\begin{aligned} L_{ij}(x) = P_j \cdot (W_j + X_{ji}^{FI} + Y_{ji}^{FI}) + (1 - P_j) \\ \cdot \Big[ [1 - \phi(x)] \cdot [X_{jj'}^{FF} + Y_{jj'}^{FF} + L_{ij'}(x+1)] \\ + \phi(x) \cdot [X_{jk}^{FC} + Y_{jk}^{FC} + H_k + X_{ki}^{CI} + Y_{ki}^{CI}] \Big], \\ j' = best(j);\ k = h(j) \end{aligned} \tag{2}$$

$L_{ij}(x)$ is the amount of delay that is incurred in the fog layer from fog node $j$ at the $x$'th time a task sent from IoT node $i$ has been offloaded. Therefore the equation for $L_{ij}(x)$ is recursive, with the parameter $x$ indicating how many times the request has been already offloaded. The base case for the recursion is $e_{\mathcal{M}}$, and the parameter $x$ begins at 0. The delay amount includes the possibilities of the request being offloaded to multiple other fog nodes, and/or sent to the cloud if necessary.

When an IoT node $i$ sends a request to its assigned fog node $j$, the request has the probability $P_j$ of being accepted into the queue, and probability $(1 - P_j)$ of being rejected. In the case where a request is accepted into a fog node's queue, it will have an average waiting time $W_j$ before it is finished being processed, and will also sustain a propagation $X_{ji}^{FI}$ and transmission $Y_{ji}^{FI}$ delay to be sent back to the IoT node that made the request. Shown below is the offloading function $\phi(x)$.

$$\phi(x) = \begin{cases} 0 & x < e_{\mathcal{M}}, \\ 1 & x = e_{\mathcal{M}}. \end{cases}$$

If the number of times a request has been offloaded in the fog layer is less than the threshold to send a request to the cloud, $\phi(x) = 0$, otherwise if the amount of times a request has been offloaded equals the threshold, $\phi(x) = 1$. Note that the second condition ensures that $x$ can never be greater than $e_{\mathcal{M}}$.

In the case where a request is rejected from entering a fog node's queue, and the offload threshold has not been reached, fog node $j$ will offload the request to its most suitable neighbor $j'$. This costs another propagation delay $X_{jj'}^{FF}$ and transmission delay $Y_{jj'}^{FF}$ for sending the request to fog node $j'$. The recursive term $L_{ij'}(x+1)$ is then added, which represents the associated costs of attempting to process the request at fog node $j'$. When a request has been offloaded the maximum number of times, if the last fog node the request visits offloads the request, it will do so to the cloud. This then requires propagation delay $X_{jk}^{FC}$ and transmission delay $Y_{jk}^{FC}$ from the last fog node to the cloud server $k$ to be sustained. There will also be a cloud processing delay $H_k$, and propagation delay $X_{ki}^{CI}$ and transmission delay $Y_{ki}^{CI}$ for sending the response back from the cloud server $k$ to the IoT node $i$.

The final pieces of the fog delay equation are the functions *best(j)* and *h(j)*. *best(j)* assigns a fog node $j$ to its best or most suitable neighbor, $j'$. *h(j)* assigns the fog node $j$ to its cloud server $k$. The value of *best(j)* may change at any time because the best neighbor for $j$ may depend on the estimated waiting times of the neighboring fog nodes.

## 3.3 Fog Offloading Simulation

To test the performance of this model, Yousefpour et. al created a simulation of a fog computing network that includes IoT devices, fog nodes, and cloud servers [11]. The network consists of five hundred IoT nodes, twenty-five fog nodes, and six cloud servers for each trial that is run. IoT nodes are capable of producing both light and heavy requests. IoT nodes are considered to have processing power comparable to an Arduino Uno R3 microcontroller, and fog nodes are considered to have processing power comparable to an Intel dual-core i7 cpu. Yousefpour et. al note that "In the worst case, a fog node's processing capability is found to be around 3000 times faster than that of an IoT node generating type light requests... and 200 times faster than that of an IoT node generating type heavy requests" [11]. Cloud servers are also one hundred times faster than fog nodes.

The network can behave in three different ways that are compared during the simulation. The first way is called No Fog Processing (NFP) in which the IoT devices in the network will either process their own requests, or send the requests directly to the cloud without using any fog nodes. The second way is called Light Fog Processing (LFP), in which IoT nodes can interact with both the fog and cloud layers of the network, but only send light requests to the fog layer. Heavy requests can still be sent to the cloud in LFP configuration. The last configuration is All Fog Processing (AFP) in which IoT nodes can send both light and heavy requests to the fog layer and to the cloud, i.e. there are no restrictions on where requests can be sent.

## 3.4 Simulation Results

Figure 1 shows the service delay among all three types of network configurations as the offload limit $e_{\mathcal{M}}$ within the fog layer is increasing. Expected results using the analytical model, denoted by the label *-anl*, are compared to the simulation results, which shows that they are nearly equiva-

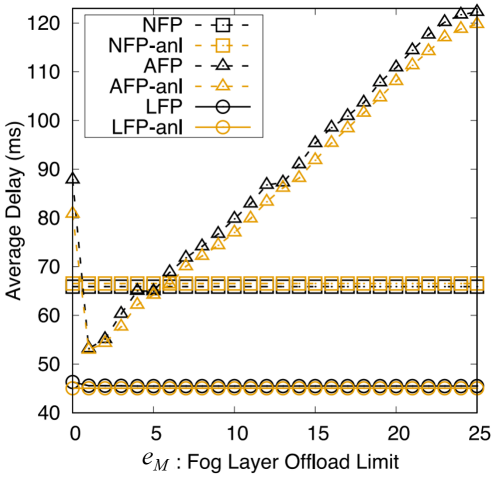


Figure 1: Effect of increasing the cloud offload threshold on service delay [11]

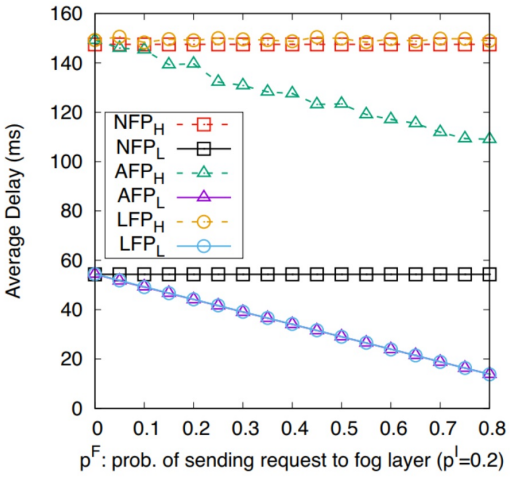


Figure 2: Effect of increasing the probability of using fog layer on service delay [11]

lent. Increasing $e_{\mathcal{M}}$ has no effect on the NFP configuration, which does not use the fog layer, however, it doesn't appear to affect the LFP configuration either. The reason for this is that the propagation and transmission delays are negligible for light requests. For the AFP configuration, it can be seen that when offloading heavy requests too many times, service delay is significantly increased. When $e_{\mathcal{M}}$ is greater than five for AFP, it is not advantageous to use the fog layer at all. These findings suggest that the optimal $e_{\mathcal{M}}$ for a fog computing network with offloading is low for heavy requests.

Figure 2 shows the service delay among each type of network configuration as the probability of sending requests to the fog layer increases. The value of $e_{\mathcal{M}}$ for this trial is 1. To give a better sense of how each configuration performs, delays for light and heavy requests are separated in the graph and are denoted by the subscripts L and H. Note that $p_i^I$ for all IoT nodes is set to 0.2. This graph shows that as requests are sent to the fog layer with increasing probability, the average service delay decreases for the requests that are able to take advantage of the fog layer based on which configuration they are sent from. Both Figures 1 and 2 demonstrate how utilizing fog computing to offload tasks results in lower service delay, and thus better quality of service for users compared to traditional cloud computing.

Note that in Figure 1, LFP mode outperforms AFP mode. This occurs because the settings used for the trial in Figure 1 inundated the fog node queues with requests, so that they were almost always full. If they were not full, the only type of requests that could enter the queues were light because the queues were still nearly full. Therefore, whenever a heavy task was sent to the fog layer, it would not be able to be accepted into any of the queues, which contributed to the increased delay seen for AFP mode.

# 4. SERVICE MIGRATION

The second technique covered in this paper for increasing the quality of service of edge networks is known as service migration. Service migration is only applicable to MEC networks, and improves service within these networks by decreasing latency. Each MEC node has a limited zone of coverage called a service area. If devices travel outside of the service area of a MEC node they are connected to, it can result in slower response times for the services they are using or even result in a suspension of service [8]. Service migration is the process of a MEC network moving the services that a device is using from one node to another based on the availability of another node with a better service area for the location of the device.

There are several factors which contribute to the decision for a MEC node to migrate its services to another node. One factor is the suitability of the current service area for the device. Another consideration is how many viable alternative nodes there are for the service to migrate to. A third is how much time and network resources it will require to perform a service migration (migration cost). Finally, nodes must try to predict if or how long a device is likely to stay within a particular service area.

One key difference between fog computing and MEC is that it is common for MEC models to use virtual machines to handle service requests [1]. MEC nodes do not always need to transfer every layer to another node when they migrate a service. Nodes often store copies of operating systems and applications when they are frequently used so that they can be reused for the instance layers that require them. In such cases, only the instance layers need to be transferred between nodes, which reduces the migration cost of performing a particular service migration. This is because less data needs to be sent over the network and MEC nodes do not have to dedicate as much time and hardware resources to downloading data.

## 4.1 Follow-Me Cloud Scheme

One of the proposed methods for conducting service migration over a MEC network is proposed by Taleb et. al in [7] called Follow-Me Cloud (FMC). FMC is a framework designed to transfer user services from one MEC node to another so that the service "follows" the user as they move to provide better quality of service. In this scheme however, the term data center (DC) is used instead of the term MEC node, but they can be considered to be roughly the same in this context [8]. In the FMC framework, depicted in Figure

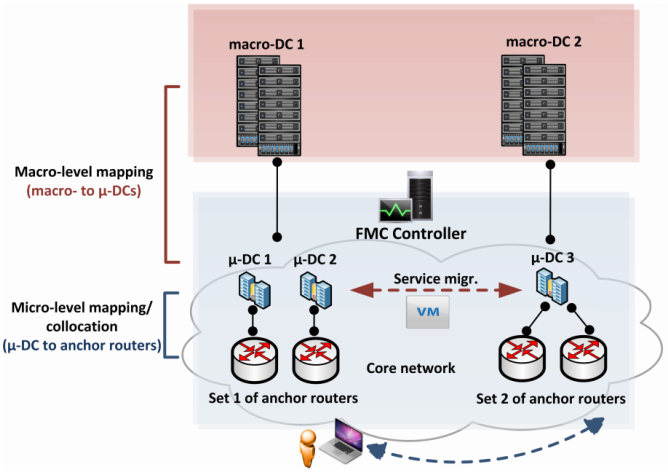


Figure 3: Model of a FMC MEC network. $\mu$-DCs stand for micro-DCs [7]

3, multiple DCs are connected to each other as a network to provide service for users in a particular geographic area. These DCs are also connected to routers, or other network connecting devices, so that devices can receive service from the network at these points [7]. The DCs implement the virtualization scheme detailed in the background section.

Each DC can be classified into two types. Micro-DCs are implemented as local or integrated storage for the routers and are capable of running VMs. They also match users with their corresponding services and conduct migrations to other micro-DCs. Macro-DCs are servers which provide more permanent storage required for VMs and instantiate services that had not been running on micro-DCs. Each macro-DC is conncected to one or more micro-DCs, and each micro-DC is connected to one or more routers. Each FMC network also contains a controller (FMCC). The FMCC is responsible for making decisions on whether a service should be migrated based on the migration cost.

## 4.2 Markov Decision Process

The Markov Decision Process (MDP) is one of several methods that can be implemented for performing service migration in MEC networks. An MDP is a mathematical construct that is used to represent situations that require making decisions which involve outcomes that are partially random, but also partially controlled [9]. An MDP uses time to be a factor over which decisions are made. A discrete time MDP considers time to be made of equal sized chunks that cannot be broken up further, whereas in a continuous time MDP, time progresses uninterrupted. The key elements of a discrete time MDP include states the system can be in, actions that can be taken at those states, transition probabilities of moving to different states given an action and a present state, and rewards for performing actions within states [9]. A continuous time MDP is similar, but the most notable difference is that instead of using transition probabilities, transition rates are used. A transition rate is how often a particular state is transitioned to when an action is performed at a different state [2].

MDPs for modeling specific networks can be used in the FMCC for making migration decisions. The goal of using

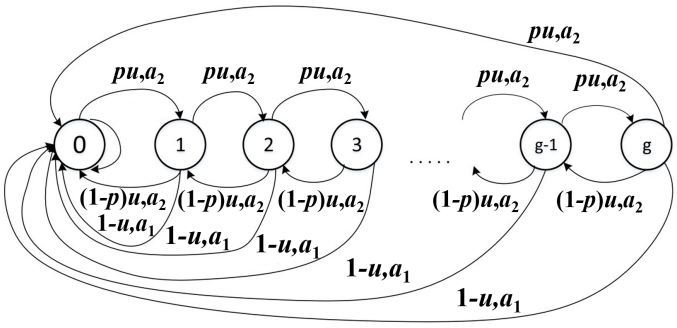


Figure 4: One dimensional MDP [8]

an MDP is to find the optimal action to perform at each state, or in other words, to maximize the expected reward for performing actions. There are two variants of MDP that can be used in the FMCC, one dimensional MDP and two dimensional MDP [8]. In one dimensional MDP, devices are considered to be moving along a straight line, with MEC nodes placed along that line and devices moving through their service areas. We will proceed to discuss the implementation of the one dimensional version.

Figure 4 shows a continuous time MDP for an MEC network, with the circles representing the states of the MDP, the arrows showing the transitions, and the notation above the arrows constituting the transition rates. Each state represents an MEC node and the numbers denote the distance away from the node that is currently processing a user's service. The circle $g$ is the node that is the maximum distance away from the current node; if the device travels farther than this distance the service must be migrated to the optimal MEC node, otherwise service interruption would occur. At each state an action can be taken, "migrate" or "don't migrate". $a_1$ is the migrate action, which means that a device's service will be migrated to the optimal MEC node. In this scheme, the optimal MEC node for a device is always the node that contains the device in its service area [8]. $a_2$ is the other action that can be taken, in which the same node continues to process a device's service. $u$ is a parameter for the mean of an exponential distribution, $1/u$, which corresponds to approximately how long devices will remain in a state (stay time). $0 \leq p \leq 1$ is the probability that a user's service will move to another MEC node further away from the node processing their service, and $(1-p)$ is the probability they will move closer to the node processing their service.

To find the best actions to take for MDPs that model MEC networks, the continuous time MDP must be converted to a discrete time MDP [8]. With this conversion, transition probabilities can be derived from the rates. The probabilities are shown below by the piece-wise function:

$$p(s'|s,a) = \begin{cases} 0 & s' = 0, a = a_1 \\ p & s' = s+1, s \neq g, a = a_2 \\ 1-p & s' = s-1, s \neq 0, a = a_2 \\ 0 & \text{otherwise.} \end{cases}$$

$s$ is a state that the user is currently in, $s'$ is the state they may transition to, and $a$ is an action. The notation on the left hand side is a conditional probability. It gives the probability of transitioning to a particular state, assuming

the device is within a specific state and that a particular action has been performed.

The function that gives the reward for performing actions involves both a cost and benefit. The cost of performing a migration can be given as the piece-wise function:

$$c(a) = \begin{cases} c_m & a = a_1 \\ 0 & a = a_2 \end{cases}$$

$c_m$ is the cost of partially or completely migrating a service. The cost involves starting the VM at the new MEC node, deciding what information is necessary from the old MEC node to be sent to the new MEC node, and sending the information over the network link between them [7]. There is no cost involved if the service is not migrated. The benefit of performing a migration is given by $Q(s')$, which is the quality of the connection to the MEC node processing a service from the perspective of a user who is currently in the service area for state $s$ [7]. Taleb et. al used latency in their simulation of FMC to measure quality. The reward function is then defined as:

$$r(s, s', a) = Q(s') - c(a)$$

In other words, the reward is the quality that can be gained by performing a migration minus the cost required for doing so. Note that the higher the latency is between a device and the MEC node processing their service, the greater the reward will be for migrating. The reward function needs to be augmented with the transition rates of the continuous time MDP shown in Figure 4 before it can be used for optimization [7]. This augmented version is shown below:

$$\mathrm{R(s', s, a)} = \mathrm{r(s', s, a)}\frac{\alpha+\beta(s',s,a)}{\alpha+c}$$

Where $\alpha$ is a predefined constant, $\beta$ is the transition rate between s and $s'$ with action $a$, and $c$ is a stay time parameter [7].
Together with the transition probabilities described earlier, the modified reward function is used in an optimization procedure called value iteration, which finds the optimal set of actions to take at each state [7].

## 4.3 FMC Simulation and Results

In order to test this scheme, Talib et. al created a simulation with two each of macro-DCs, micro-DCs, wireless routers, and one FMCC [7]. The macro-DCs run Windows XP VMs. In the simulation, two clients are connected to one of the wireless routers, and in both cases their service is being run on the DC that is further away from them. The network delay (latency) to reach this DC is set to be 50ms. The other DC is closer, and is therefore optimal from a client perspective for the service to be migrated to. After approximately thirty seconds of running the simulation, the service for one of the clients is migrated. For a couple seconds, the latency this client experiences spikes to 120ms, before dropping to below 10ms for the rest of the simulation. The latency for the other client remains unchanged for the duration of the simulation at 50ms. Importantly, the service for either client is never interrupted during the migration. The simulation showed that the client that had their service migrated experienced significant improvement in their ping, which corresponds with higher quality of service.

# 5. CONCLUSION

The continued proliferation of the Internet of Things is generating massive amounts of data. New networking technologies and paradigms, such as edge computing, have shown promising results in being able to help cope with this trend. In this paper, we have discussed two different kinds of edge computing, fog computing and mobile edge computing, and examined one technique for each that is designed to improve their efficiency. Being able to improve the performance of these networks has a direct impact on the quality of service that users perceive when connected to them. The refinement of these systems can help make our lives more convenient and may speed up the development of future products and technologies.

# 6. ACKNOWLEDGEMENTS

I would like to thank my adviser KK Lamberty, my instructor for Senior Seminar Elena Machkasova, and my alumni reviewer Dan Frazier for their helpful time and feedback they have provided for this paper.

# 7. REFERENCES

[1] K. Dolui and S. K. Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. *2017 Global Internet of Things Summit (GIoTS)*, August 2017.

[2] J. Linssen. Continuous-time Markov Decision Processes. Retrieved November 23, 2018 from `https://dspace.library.uu.nl/bitstream/1874/336084/2/bachelor_thesis_linssen.pdf`.

[3] J. Ni, K. Zhang, X. Lin, and X. Shen. Securing Fog Computing for Internet of Things Applications: Challenges and Solutions. *IEEE Communications Surveys Tutorials*, 2018.

[4] U. of California Berkeley. Packet delay, cs168. Retrieved November 23, 2018 from `https://inst.eecs.berkeley.edu/~cs168/fa14/discussion/slides1.pdf`.

[5] G. M. Schneider and J. L. Gersting. *Invitation to Computer Science, 6th Edition*. Cengage Learning, 2013.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 2016.

[7] T. Taleb, A. Ksentini, and P. Frangoudis. Follow-Me Cloud: When Cloud Services Follow Mobile Users. *IEEE Transactions on Cloud Computing*, February 2016.

[8] S. Wang, J. Xu, N. Zhang, and Y. Liu. A Survey on Service Migration in Mobile Edge. *IEEE Access*, 6:23511–23528, April 2018.

[9] Wikipedia. Markov decision process. Retrieved October 13, 2018 from `https://en.wikipedia.org/wiki/Markov_decision_process#Continuous-time_Markov_decision_process`.

[10] Wikipedia. Virtual machine. Retrieved October 13, 2018 from `https://en.wikipedia.org/wiki/Virtual_machine`.

[11] A. Yousefpour, G. Ishigaki, R. Gour, and J. P. Jue. On Reducing IoT Service Delay via Fog Offloading. *IEEE Internet of Things Journal*, 5:998–1010, April 2018.