National College of Ireland

Geo-Context Aware Routing Using Serverless Computing

MSc Research Project
MSc in Cloud Computing

Akash Verma
Student ID: X21128863

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: …….…… Akash Verma …….

Student ID: ……X21128863……

Programme: ……MSc in Cloud Computing… **Year:** 2022

Module: ……Research Project…….……..

Supervisor: ……Vikas Sahni…….……

Submission Due Date: ……15 Dec, 2022……

Project Title: Geo-Context aware routing using Serverless Computing…

Word Count: ……4554…… **Page Count**……18…….……..

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……

Date: ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Geo-Context Aware Routing Using Serverless Computing

Akash Verma
X21128863

Abstract

When a vehicle is in motion the change in tire pressure, speed, engine temperature, and other vehicle parameters can help understand the maintenance required, vehicle condition, driving style, etc. This data when processed in a cloud data center consumes more network. This also results in latency, which is a problem for real-time applications.

The shift towards autonomous vehicles is facing latency as a major blocker. This problem can be solved by moving the computing resource closer to the end devices. The service can be placed at a location that is preferably at the edge or close to the edge of the network instead of remotely. The dynamicity of topology for the ecosystem also results in ad hoc demands to serve the payloads on computing resources. The service placement also governs QoS and latency. It is a challenge to identify the node with the least latency, which has the service deployed and running.

Considering the latency requirement and dynamicity of the network, this paper exploits the serverless computing model and presents an architecture to serve IoT devices with the nearest possible Fog/Edge server which has the service running.

1 Introduction

With the ever-increasing number of devices connected to the internet, human-to-device or device-to-device interaction produces a massive amount of data. There are billions of such devices and is projected to increase three folds after the evolution of IoT[1]. The modern vehicle produces enormous data. The telemetry data is produced by the sensors continuously. It can help in decision-making like real-time traffic updates, predictive maintenance, driving style prediction, etc. If processed at a data center, it takes multiple hops and may result in latency.

The answer to this question is fog computing, which was proposed by Cisco Systems in 2012. It emphasizes placing the computing power close to the source of data (Bonomi, et al. 2012.). But it is challenging, when the data source is moving, this results in a continuously changing network topology, as it happens in the vehicular network. Modern vehicles are equipped with a smart onboard device or OBD, which is connected to the mobile base station. These base stations serve as an edge node, The scenario when the computing is performed at the base station to which the IoT device is connected using radio access network is termed mobile edge computing or multi-access edge computing (Giust et al; 2018.). With the rollout of the 5G network, evolution of SDN, use of CRAN and VNF, with the abstraction over the network hardware, the services can be deployed on a virtual machine at the base station (Khan et.al; 2018), but there is a limit to the computing power available at the base stations.

[1] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

However, if not at the edge, the service can also be deployed at any hierarchy of the network and closer to the edge (Carnevale et al;2018). These nodes are called fog nodes.

When there is a rise in demand for fog services and limited resources are available, the need to optimize the use of resources is crucial. Thus, the deployment strategy needs to be dynamic to serve as many applications as possible in a range of time.
The service placement can be based on a different strategy and classified as Exact, approximations, heuristic, and meta-heuristic (Salaht, et al; 2020*)*. The Orchestration of service is the scope of the research once it is deployed on different edge or fog servers.
This research intends to implement the architecture that is based on REST principles, where each instance of deployed services has an endpoint, and the connection to these endpoints is made by IoT devices. The discovery of the nearest telemetry processing service node either at edge or fog is done by the geolocator service, which uses geographic location to detect the proximity of the vehicle. It assumes that geographically nearest deployed service should be closer to the vehicle in the network. The complete implementation is done in a serverless way, which is independent of the infrastructure and the components are containerized. They can be deployed from a server-grade machine to microcomputers like the raspberry pi.
To solve the problem, the paper investigates the existing related work done in the field and the approach used. Under the research methodology, the environment for simulation is set, where the vehicle is simulated to run on a road picked from the map. The telemetry processing units are placed enroute. The architecture is designed such that the vehicle run, and the telemetry service coverage area are given as the input configurations to the system. Various components, their functions, and salient features are discussed in the design specification section. In the implementation section, the algorithm used is depicted. Once these components are set up and programmed, simulation is performed on the Ireland M1 motorway configuring 6 local TPUs and one cloud TPU, the results obtained are then critically analysed and findings are shared in the evaluation section. Finally, the paper talks about the conclusion and future scope of the research.

2 Related Work

2.1 The Edge and Fog Ecosystem

The conventional data center centric client-server architecture is affected by BLURS viz. bandwidth, latency, uninterrupted, resource constraint, and security (Buyya and Srirama; 2019). Fog and edge computing can address the BLURs of clouds of things. The three-layered architecture of the cloud fog ecosystem is termed as a possible solution (Yousefpour et.al; 2019).
Hamdan S. Et. al. classified fog and edge computing architecture styles into five categories namely, data placement, big data analysis based, security-based, machine learning-based, and Orchestration based architecture (Hamdan et al; 2020). The architecture that is implemented in the paper falls under the orchestration-based architecture, various edge and cloud architecture was analyzed as a part of the literature survey.

2.2 Multi-Access Edge Computing and SDN

MEC is the key enabler for the internet of things, it helps bring computing power to the edge of the network. It opens a wide range of possibilities for computing requirements when the smart device is in motion. The main advantage of using IoT along with MEC is the reduced network traffic, latency, and scaled network service. (Taleb et al; 2017). With the roll-out of 5G, the enormous bandwidth is accessible to connected devices, and the Automotive IoT can realize real-time traffic monitoring (Balid, et.al; 2018), and other high bandwidth services like infotainment applications.

The SDN or software-defined network is one of the features of 5g, where the network can be programmed as software. This concept was introduced around open flow at Stanford University. This provides agility to the underlying infrastructure.

2.3 VANET VISAGE and FSDN

Different research was conducted about possible fog-edge architecture for vehicle networks. vehicle ad hoc network (VANET) a concept derived from the mobile ad-hoc network (MANET) has gained popularity and much research is going around it. The VANET-type architecture proposed by Nyugen et.al; called FSDN proposed installing roadside units (RSU), to keep the vehicle connected. The vehicle transmits the data to the RSU, which has the required processing power to process the data. The main distinguishing feature of the architecture was the use of varying technology for communication. Vehicle-to-vehicle and vehicle-to-infrastructure communication is done using a wireless connection or WAVE. The vehicle to the base station was done using WiMAX, 3G, and 4G. All the communication with RSU and RSUC was done using high-speed broadband (Truong et. al; 2015).

The VISAGE (Vehicular Infrastructure SdN Approach and a foG computing architEcture) was proposed with the evolution of the 5G network, it is also termed 5G VANET. The architecture is the combination of a multilevel SDN-based approach that uses vehicles as infrastructure, to deal with communication and computational capability. The offloading is done to the vehicles connected through VANET. The proposal demonstrated The benefits in terms of communication and computation capacities. (Soua and Tohme; 2018).

2.4 Other Architectures

M. Verma et. al. proposed an architecture that used a real-time efficient scheduling algorithm for load balancing in a fog computing environment where the edge and fog node acts as a cache for the service requested by the smart device. If the service was available the request was served, or else the request was forwarded to the cloud. This design had a fallback mechanism. Each time when the service was not available, the time required to serve a request was higher than the time to serve it directly from the cloud. The service placement is one of the major blockers for fog and edge computing when a large area is considered and there is veracity in the service offered. There is a high chance of a cache miss when multiple services are on demand, and resources are limited (Verma et al; 2016).

Habak et. al. proposed an architecture that leverages mobile devices to work as a fog node to provide the desired service. The paper presented a Femto cloud architecture showing its efficiency and ability. It was a novel approach to use the resources available in the crowd based on cooperation. It is difficult to use this kind of architecture that is community-based for commercial applications. Whereas, It can be applied to develop social networking applications like streaming the shared content for a specific time, only when the topology of the network does not change. The consent of the user would be a major blocker in such architecture (K. Habak et.al; 2015).

Sun and Ansari Proposed an architecture, which used a proxy VM to collect the metadata, analyze the stream, and transmit it to the application serving VM. The paper also proposed the modification of the base station to support different communication protocols used in IoT like Zigbee, Bluetooth low energy, MIMO, low-power wide-area technologies, and narrow-band IoT communications (X. Sun and N. Ansari; 2016). The modification to the base station proposed in the research had an impact on existing infrastructure.

Okay and Ozdemir also proposed a hierarchical fog-based architecture, where fog and cloud controllers are responsible for taking actions on local frequent events and global rare events respectively, which reduced the data transmission overhead and routing delay (Okay and Ozdemir; 2018).

Gupta et.al. proposed an architecture that was a microservice-based software-defined network, which introduces service-oriented middleware and exploits VNF to improve the user experience. The simulation was performed on the Mininet emulator platform that emulates a real scenario by creating virtual instances of network devices. The finding concludes that QoS-aware routing delivered a better data delivery rate. The simulation was performed for a smart home system with a patient monitored by the sensor for emergency scenarios (Gupta et.al; 2016). Table 1 depicts the summary of related work done.

Table 1: Summary of related work

Paper	**Concept**	**Implementation**
FSDN	Proposed V2V, and V2I communication using a centralized SDN.	No simulation / Implementation was done. Scope for future work
VISAGE	Proposed the use of the vehicle as infrastructure, leveraging 5G.	No simulation / Implementation was done. Scope for future work
RTES	Introduced a scheduling policy called real-time efficient scheduling (RTES) load balancing algorithm in fog infrastructure.	CloudSim tool was used for simulation.
Femto Clouds: Leveraging Mobile Devices to Provide	Use of mobile devices as infrastructure using an android	Implemented an android application that allows the user

Cloud Service at the Edge	application governing the data sharing policy.	to enter preferences for resource-sharing policies.
EdgeIoT: Mobile Edge Computing for the Internet of Things	Introduced a Hierarchal infrastructure transmitting data from IoT devices from proxy VM to application VM.	The mathematical simulation was done based on data traces of more than 13000 users in Heilongjiang Province, China. The whole area contains 5962 BSs.
Routing in Fog-Enabled IoT Platforms	Introduced a Hierarchal architecture, frequently occurring events handled by fog controller and rare events by cloud controller,	Simulated in Mininet containing Virtual hosts, switches, and controllers deployed on a single OS Python was used to develop SDN controller.
SDFog: A software-defined computing architecture for QoS-aware service orchestration over edge devices	An integrated system was developed on Service-oriented architecture.	The Mininet emulator was used to simulate network devices. It was performed for a smart home system with a patient monitored by the sensor for emergency scenarios.

3 Research Methodology

The initial setup for the simulation included two main components. The first was the route on which the vehicle trip was simulated, and the second was the telemetry processor unit and its service area.

For the first part SUMO (Simulation of Urban Mobility) toolkit was used to simulate vehicle movement. The following steps were taken to set up the mobility of the vehicle:

1. The OSM (Open Street map) was used to export the area under study, The M1 route from Dublin airport to Drogheda was the subject of simulation.
2. The exported .osm file was then used to generate random trips using SUMO.
3. The random trip was exported in XML format. These trips were based on the coordinate system
4. The geo-location transformation was applied to the XML file. It was used as an input to the time-triggered azure function that simulates the vehicle on-board unit (OBU).

For the second part of the setup telemetry processing units (TPUs) were simulated to be deployed on the M1 route. The polyline tool was then used to set the coverage or the serving polygon of the respective telemetry processing unit. The serving polygon used for the setup of local TPU is depicted in table 2.

A central telemetry processing unit was also deployed on the cloud to serve when a vehicle was in an area where none of the local telemetry processing units had its coverage.

Table 2: serving polygon for TPU

TPU Name	**Served area(figure1)**	**Served Polygon**

TPU1	S1	[-6.2845917,53.4290679],[-6.1767884,53.4229273],[-6.1692353,53.4536215],[-6.2804718,53.4618029],[-6.2845917,53.4290679]
TPU2	S2	[-6.1761017,53.4752987],[-6.1822815,53.4916515],[-6.2372132,53.4875639],[-6.2337799,53.4708006],[-6.1761017,53.4752987]
TPU3	S3	[-6.1740418,53.5227045],[-6.1781616,53.5667933],[-6.2447663,53.5598565],[-6.2344666,53.5182114],[-6.1740418,53.5227045]
TPU4	S4	[-6.1733551,53.4544397],[-6.1761017,53.4781609],[-6.2365265,53.4757076],[-6.2310334,53.4528033],[-6.1733551,53.4544397]
TPU5	S5	[-6.2461396,53.5598565],[-6.1905213,53.5623049],[-6.2063141,53.6071677],[-6.2523194,53.6026836],[-6.2461396,53.5598565]
TPU6	S6	[-6.2042542,53.6026836],[-6.2145539,53.6332479],[-6.2687989,53.6279516],[-6.2550659,53.5977913],[-6.2042542,53.6026836]

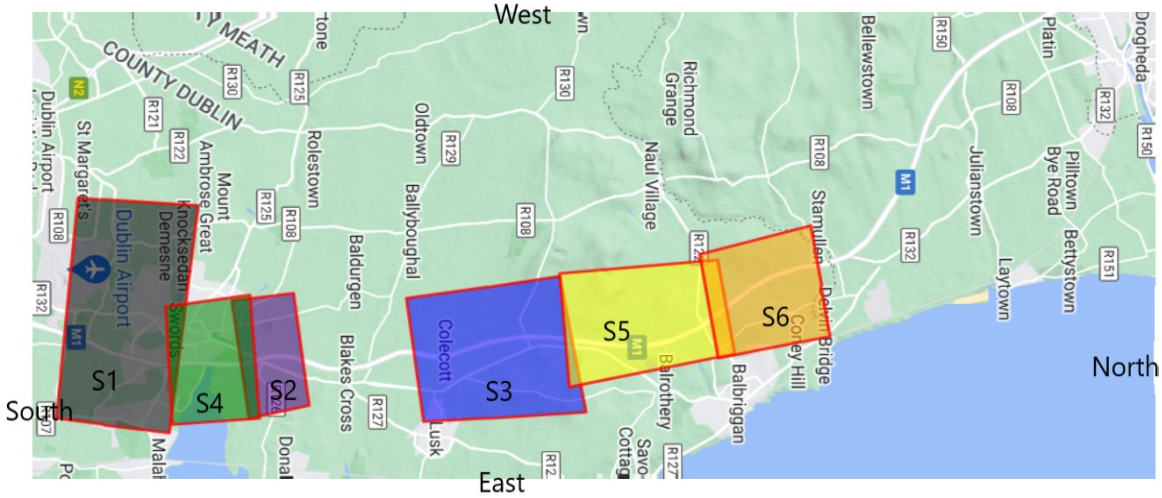

Figure 1: Simulation setup: M1 motorway, Dublin airport to Drogheda, with TPU setup.

The azure function was used to simulate the running vehicle using the workflow stated in figure 2.

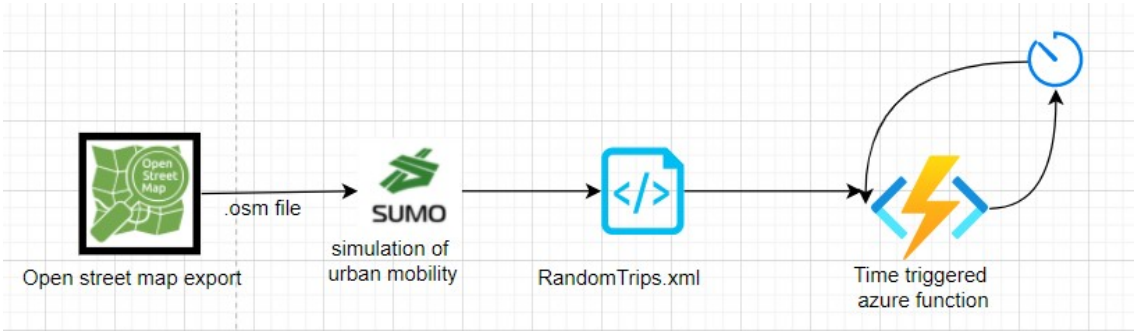

Figure 2: Simulation of moving vehicle.

Example for the dry run: Assuming the trip starts from Dublin airport to Drogheda, service should be provided as shown in order TPU1, TPU4, TPU2, TPU3, TPU5, and TPU6 as depicted in figure 3.

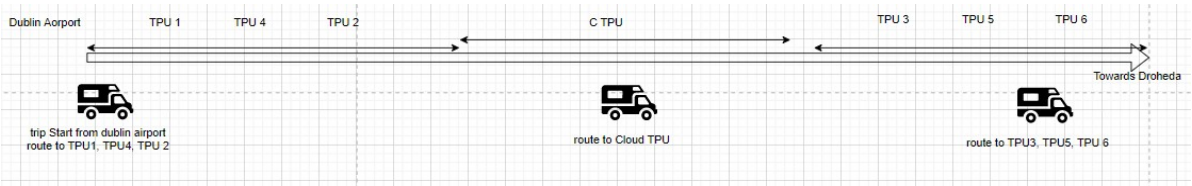

Figure 3: An example trip and serving TPU from Dublin airport towards Drogheda

4 Design Specification

The architecture is a REST-based architecture, and the components were implemented as a serverless solution. Figure 4 explains the architecture of the project. There were four main components of the architecture:

1. The On-Bord Unit (OBU): This is the simulated smart device fitted on the vehicle, it has an inbuilt GPS, to geo-locate the vehicle. It also sends the data like engine temperature, tire pressure, tilt, etc. The implementation of OBU is a time-triggered azure function, which sends data to the available TPU in every 1 second. Each OBU gets registered whenever a vehicle starts the trip, by sending the vehicle's live payload to the available TPU. The OBU was simulated as depicted in figure 2. It is also regarded as On-Board Device (OBD)
2. Telemetry Processing Unit: A Telemetry processing unit is the receptor of telemetry data; each telemetry unit is registered along with the service endpoint and geographic polygon or the coverage area. A TPU can be either a local TPU or a cloud TPU. The TPU was also implemented as an Azure function with webhook as input bindings.
3. Geo Locator Service: This was implemented as a RESTful service, which helps the OBU identify, which TPU has coverage in the current area and up to what geo-location based on the polygon set for the TPU. Once the coverage area is crossed by the OBU, new telemetry processor details were requested by OBU from the geolocator service.
4. Cosmos Database: It was used by the geolocator service to record all the data, such as registered TPUs, OBUs, telemetry events, etc. It also processes the location data as geospatial and geo JSON, to identify the serviceable TPU.

The following are the salient features of the architecture:

1. **Microservices-based architecture**: The architecture is a microservices-based architecture, where each service has its scope and boundaries. The components communicate with each other using REST protocol over the HTTP channel.
2. **Serverless**: All the component for the proposed architecture is serverless, the serverless infrastructure suits the requirement, as these services are dynamic, and may be placed on a cloudlet/edge/fog node as and when required based on parameters like priority of the service, number of users requesting the service, available capacity at the node, etc. The service provider is charged by the cloud vendor based on the time infrastructure is being used.
3. **Plug and Play**: The architecture is plug-and-play, with a provision to add more TPUs in the runtime, given the minimum configuration is done regarding the serving area

for the TPU. The vehicle with configured Geo-Locator service endpoint can be added to the service, using the vehicle live event.

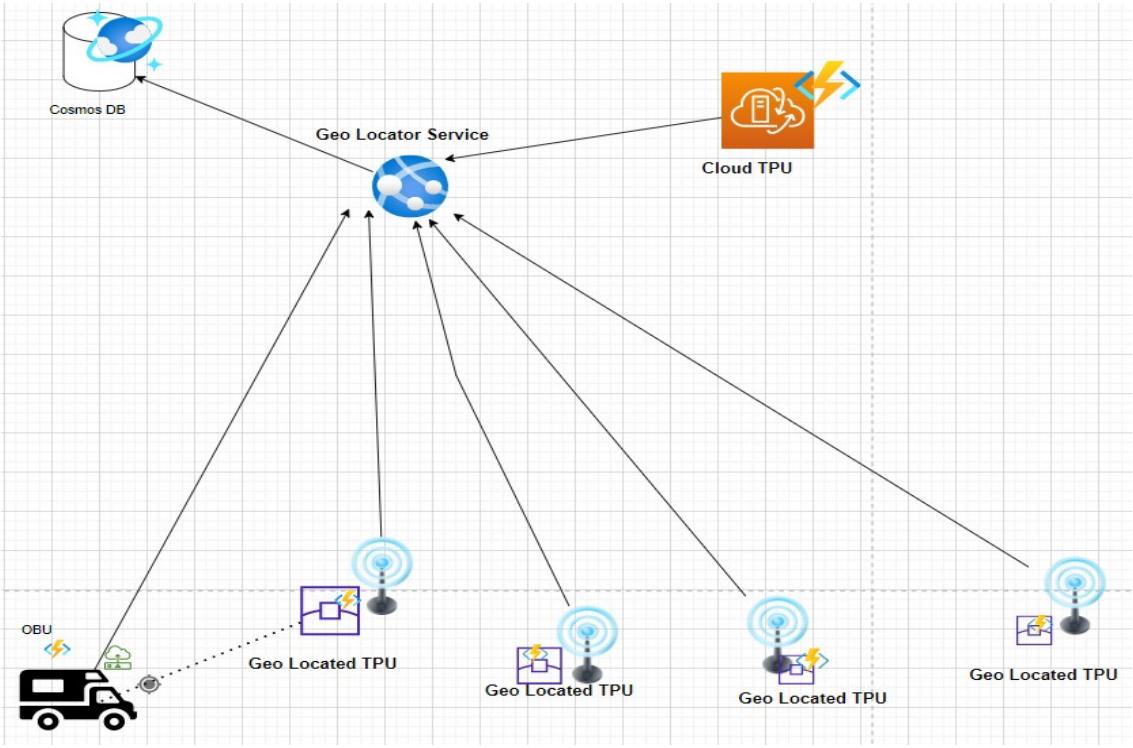

Figure 4: Architecture diagram

4. **Scalable**: Scalability is built in for the azure function, scaling horizontally, and descaling can be done based on rules specified by the administrator at a specific deployment.
5. **Geo Aware**: The telemetry processing unit that is deployed has a specific coverage, which is set as a configuration parameter, whenever a vehicle with service-supported OBU is passing by the coverage, the routing of telemetry data is done to the TPU.
6. **Fall Back**: Whenever there is no coverage, the routing is fallen back to send telemetry data to the central TPU, which is hosted on the cloud.
7. **No Vendor Lock-In**: it is possible to containerize the azure function code, such that run time is provided by the container, using dot net core image and this container can be deployed on any cloud provider's virtual machine.

5 Implementation

To implement the design, the following components were used and deployed at respective locations as mentioned in table 3:

Table 3: Component and technical stack used

Component	Technology used	Location deployed	Supporting Component
OBU	Azure Function, DotNet 6.0, NetTopologySuite	Local Machine	SUMO for vehicle trip simulation
TPU	Azure Function, DotNet 6.0	Cloud and local machine. Local deployment using the containerized image using Visual studio. Or can use .net runtime for the Azure function.	Open Street Map for determining the coverage of TPU
GeoLocator Service	Dot Net Web App (API), DotNet 6.0,	Cloud (as this is being called by both local and cloud TPUs)	NA
Database	Cosmos DB	Cloud	NA

5.1 The On-Board Unit:

Two main tasks were performed by the OBU. The first was to register itself when the vehicle was live and the second was to send telemetry data to available TPUs serving the geographic location. When the serving TPU was local, the OBU sends the telemetry info till the vehicle was in its coverage. If it was connected to the cloud, the OBU looks for local TPU before sending the telemetry data. Sending the data to a cloud TPU comes with the overhead of searching for the availability of a local TPU.

Following is the pseudo-code for implemented OBU:

```
OBU
{
        OBU(){
                VehicleLive( vehicleId, Type, location);
                Tpu = GeolocatorService .GetTelemetryProcessor(location)
        }
        Run( timer)
        {
        While(timer < tripCompletionTime)
        {
                If ( ! location within Tpu.ServedPolygon || Tpu.Type == 'cloud') ;
                  Tpu = GeolocatorService.GetTelemetryProcessor(location)
        Stopwatch.start()
        SendTelemetryData(Tpu);
         elapsedTime = Stopwatch.stop()
         GeolocatorService.Logger.Log( Tpu.endpoint, Tpu.Type, elapsedTime)
                }
        }
}
```

For the sake of analysis, one more responsibility was added to OBU, logging the time elapsed(turnaround time) in serving the request by the connected TPU.

$$T_{at} = T_{processing} + T_{latency}$$

5.2 The Telemetry Processing Unit:

A TPU also has two responsibilities, the first one is to register itself when started post-deployment, and the second i to receive the telemetry data. The TPU can be enhanced to do the data processing, to determine current traffic, possible jam, broken vehicle, failure prediction, etc, which was not in the scope of this research.
Below is the pseudo-code for implemented TPU:

```
TPU
{
  StartProcessor()
        {
                GeolocatorService.RegisterTPU( servesPolygon, Host.url, type )
        }
 ProcessTelemetryData()
        {
                LogTelemetryData()
        }
}
```

5.3 The Geo-Locator Service:

The Geo-Locator service was developed as the heart of the framework. Its core functionality is finding the serving TPU for the vehicle. The service also had a data access layer that connects to the database to find the serving TPU. Calling the Geo-Locator service was as costly as calling the cloud TPU, as it was deployed on the cloud data center. The architecture was designed such that, the service was called only when TPU was switched, due to no coverage, or the OBU was connected to a cloud TPU and it is now looking for a local TPU. The registration of the OBU and TPU was also done by the Geo-Locator service.
Following is the pseudo-code for implemented Geo-Locator Service:

```
GeoLocatorService
{
  RegisterTPU(tpu)
        {
                Database.SaveOrEdit(tpu)
        }
RegisterOBU(obu)
        {
                Database.SaveOrEdit(obu)
```

```
        }
GetServingTpu(lat, long)
        {
                vehiclePoint = Point(lat, long)
                 tpu =Database.TpuContainer.where(vehiclePoint.Within(tpu.servedPolygon))
                if (tpu == null) tpu = Database.TpuContainer .cloudTPU
                return tpu;
        }
LogResponseTime(tpu, elspsedTime)
        {
                Database.LoggingContaner.Log(tpu, elspasedTime)
        }
}
```

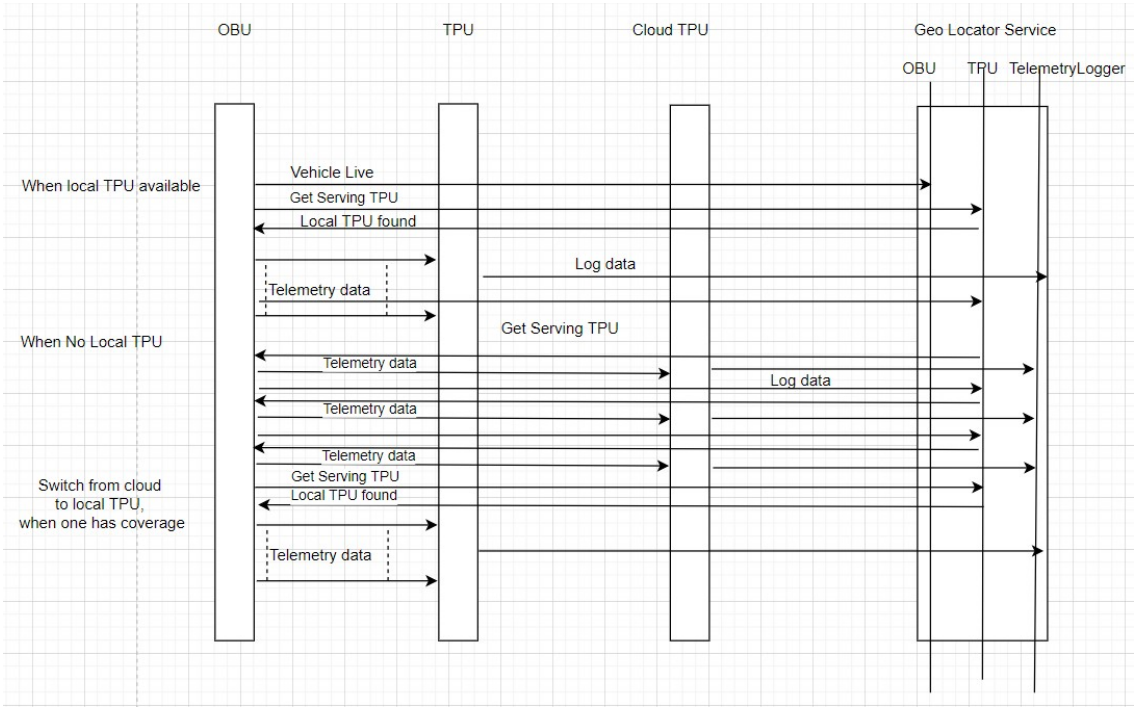

Figure 5: Event flow Diagram

As depicted in figure 5, three possible workflows were executed by the framework:

1. Routing to local TPU: In this scenario when the vehicle goes live, the Geo-Locator service returns a local TPU. All the telemetry data is sent to this TPU up till its coverage. In this case, only one packet is routed to the cloud which is the call to the Geo-Locator service for GetServingTpu().
2. Routing to Cloud TPU: The second flow in figure 5 is the scenario when no local TPU is available. For each telemetry event raised, there are two calls, that are made to the cloud. The first is for sending the telemetry data, and the second is for checking if a local TPU is available at that time in a given geography.

3. Switching: The third scenario is the transition scenario where, the local TPU is found, before sending the telemetry data to the cloud. It also involves a single call to the cloud, and all other calls are made to local TPU regarding the telemetry events.

6 Evaluation

The experiment was performed with the OBU running on the local machine with six TPUs deployed as a container in the local machine (Isolated environment). A cloud TPU was deployed in the north Europe region(Dublin data center) for Microsoft Azure cloud. The OBU was simulated to move randomly on the M1 motorway for 780 seconds, each second stepping the OBU to a new geo coordinate in route obtained by SUMO random trip traces XML file.

6.1 Case Study 1 – Finding outliers

On analysis of the graph for timestamp vs turnaround time, some turnaround time created the peak. These were the result of function cold startup or other environmental conditions. These data points were excluded from the analysis and were considered outliers.

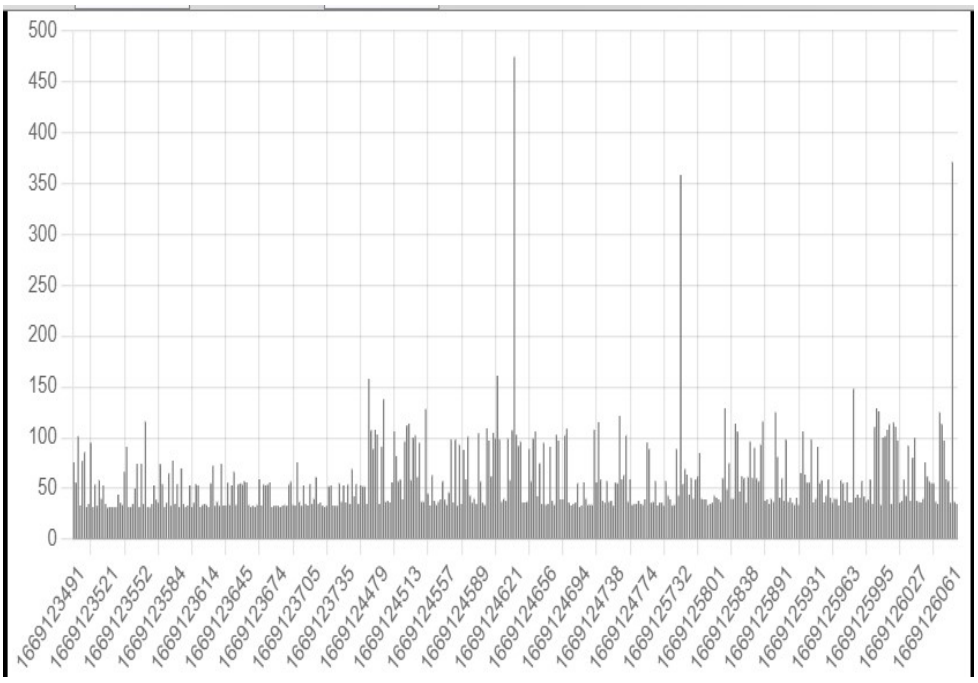

Figure 6a: Timestamp vs turnaround time for local TPU

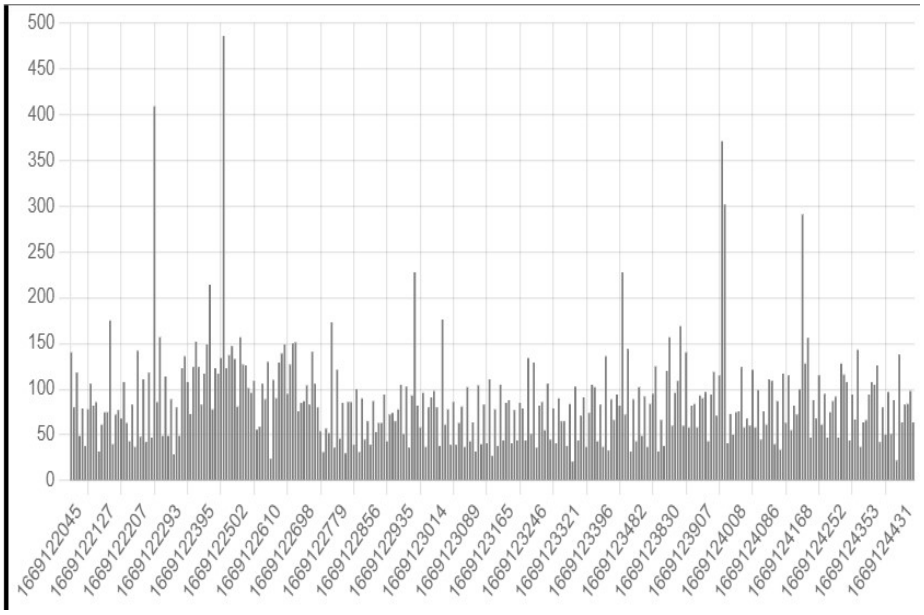

Figure 6b: Timestamp vs turnaround time for cloud TPU

Observation: The outliers as shown in Figures 6a, and 6b were filtered before analyzing the TPU's turnaround time. There were three instances where the turnaround time is greater than 350 ms for local TPUs, the same as the cloud TPUs. Only the records under the value of 350 ms were considered as a part of the turnaround time analysis.

6.2 Case Study 2 - Average turnaround time by node type

The average turnaround time was found after grouping based on the type of TPUs for the simulated run. Figure 7 shows the average turnaround type vs of TPU.

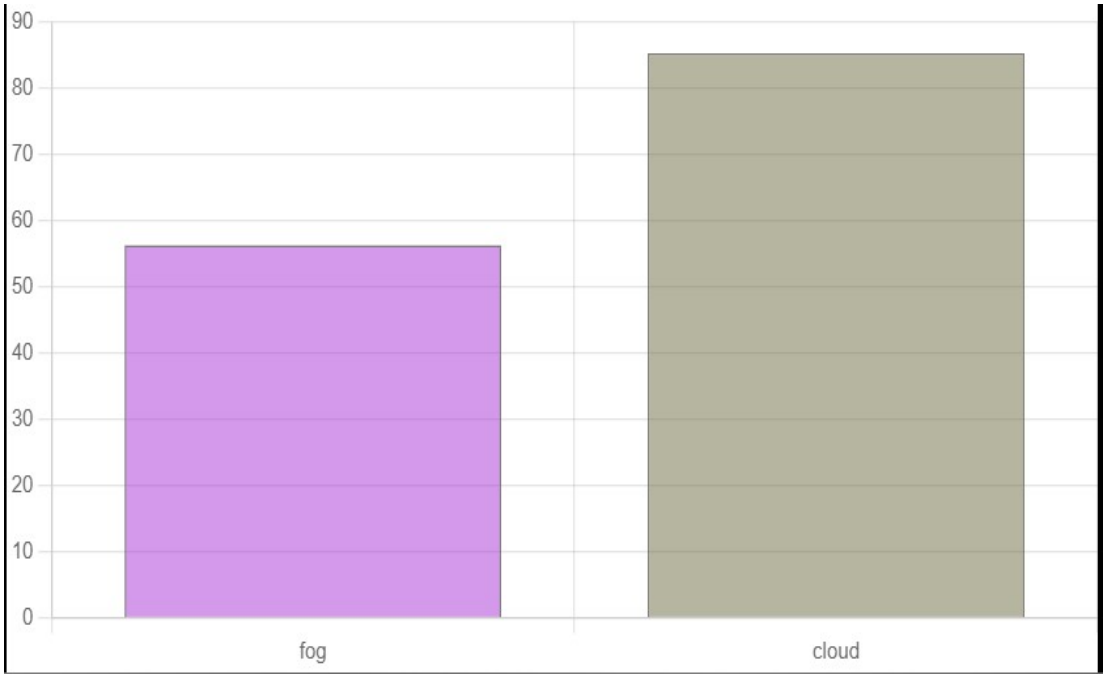

Figure 7: Average turnaround vs type of TPU

Observation: The average turnaround time for the local TPU was recorded to be 56.269 ms, whereas for cloud TPU it was 85.272 ms. On average the turnaround time for the fog node is 0.6599 times less than the cloud node.

6.3 Case Study 3 – Top 25 fastest turnaround time

The top 25 response turnaround time was analysed to find the occurrence of fog and cloud instances for the simulated run. Figure 8 shows the Top 25 TPU types vs time in ascending order

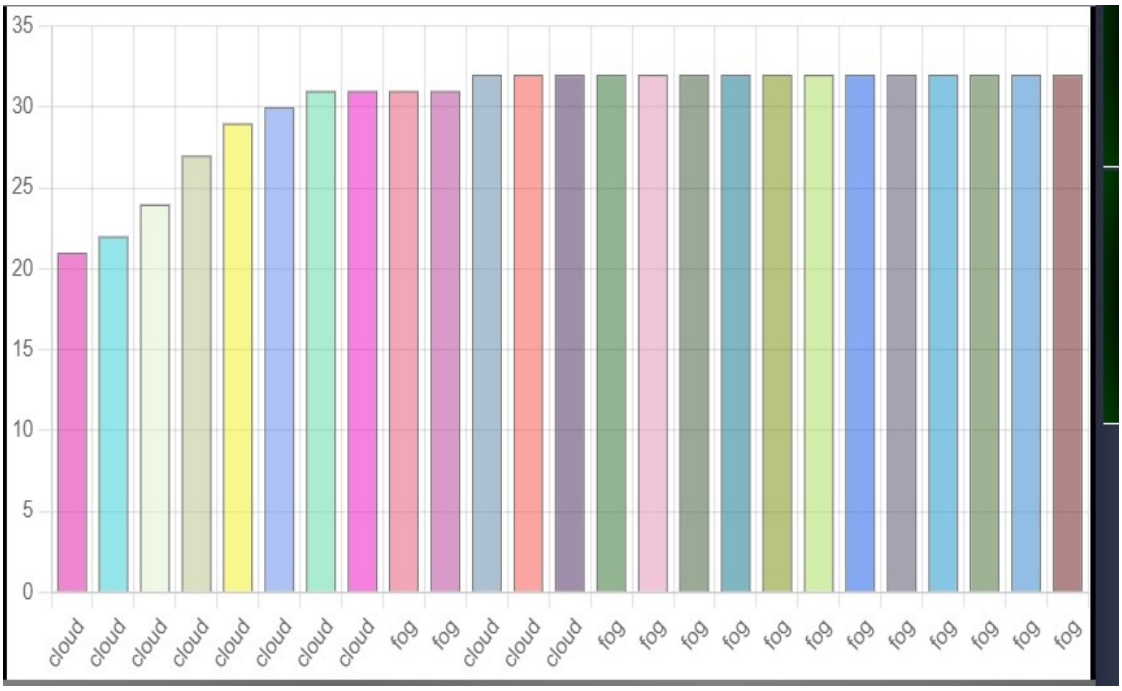

Figure 8: Top 25 fastest-served instances

Observation: When the TPU logs were analyzed for the top 25 fastest-served requests, the ratio of fog to the cloud was 14:11. There were 14 fog instances and 11 cloud instances. There were very few instances when the cloud TPU performed exceptionally better than the fog but the fog had a consistent turnaround time and analysis is done on the response time).

6.4 Case Study 4 – Top 25 slowest turnaround time

The top 25 response turnaround time was analysed to find the occurrence of fog and cloud instances for the simulated run. Figure 9 shows the Top 25 TPU types vs time in descending order

Observation: In the 25 top slowest-performing occurrences, the local TPU made only three occurrences, while 22 occurrences were of cloud TPU. The highest time taken is 300ms. After 9 occurrences of cloud turnaround time, the highest time taken by the fog node is close to 280 ms.

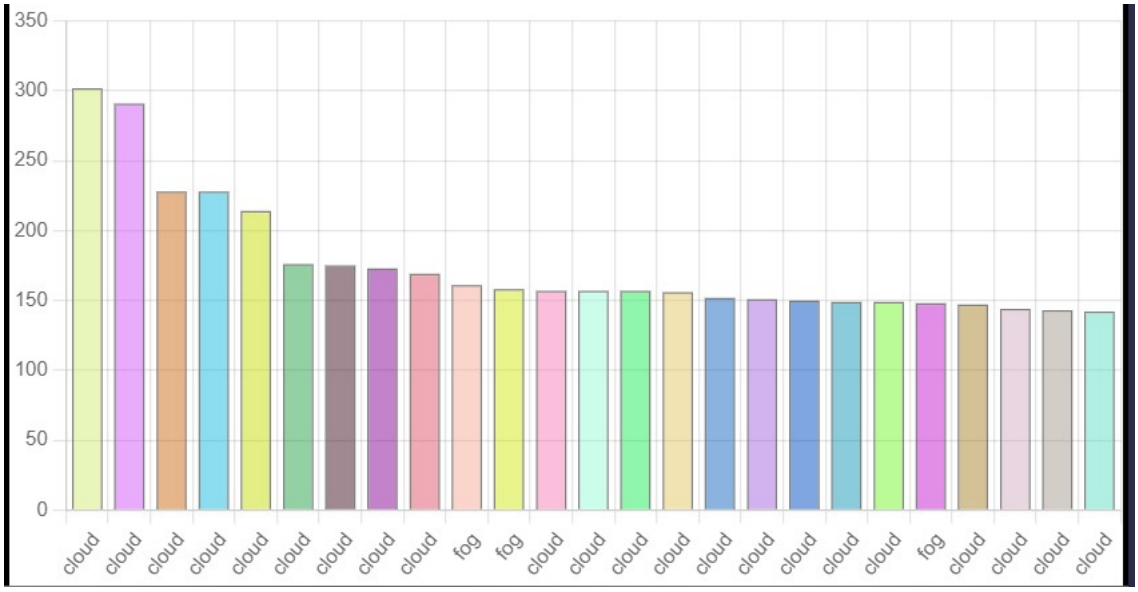

Figure 9: top 25 slowest-served instances

6.5 Case Study 5 – Cloud TPU in current region vs remote region

In a separate run, a new cloud TPU was deployed at azure central India region, with an existing TPU in the north Europe region. The comparison on average turnaround time was made with the OBU running locally connected to a broadband network in Dublin. Figure 10 shows the plot for CloudTpu grouped by endpoint vs turnaround time.

Figure 10: average latency when cloud TPU north Europe vs central India.

Observation: central India being remotely located from Dublin, the turn around time was significantly high. In figure 10, the average latency for central India TPU was around 690 ms, and for northern Europe, it was 120 ms. The location had a very high impact on turnaround time.

6.6 Discussion

Azure function cold start-up, environment factors like hardware available and core network infrastructure impact the total turnaround time for a request. The experiment is done on a free tier hosted azure function at the cloud for cloud TPU and docker containers limited to 1Gb of memory locally for fog TPUs. The idea behind the experiment was to do a real-time comparative analysis of the implemented architecture and the impact of hosting the service nearer to the data source. The case study in the evaluation proves that the frequency of the local telemetry node responding faster than the cloud telemetry node is higher. Within the cloud telemetry nodes, the response time also depends on the difference in the region between the data source and services.

7 Conclusion and Future Work

Considering the observation, when the local TPU nodes were running in isolation, on docker containers they performed faster than cloud TPU. During the research, the local TPUs were also observed to be performing slower than cloud TPU consistently, when they were executed in a shared environment, using .net core azure function run time on a single machine (not on isolated containers). There is a possibility that the cloud TPU is performing better than a fog TPU, due to environmental conditions like load, network outage, queue, storage, past performance, etc. The research can be extended to route the data, based on the prediction made by a machine learning algorithm and the data can be routed to the cloud in rare scenarios, despite local TPU coverage to enhance the performance of the architecture.
The architecture can be decoupled using an event grid or Kafka instead of sending payload directly to the service, via the use of events.

Link to video presentation: https://web.microsoftstream.com/video/240ab4dd-3a22-4181-a866-d885cb6a2eae

References

Balid, W., Tafish, H. and Refai, H.H., 2017. Intelligent vehicle counting and classification sensor for real-time traffic surveillance. *IEEE Transactions on Intelligent Transportation Systems*, *19*(6), pp.1784-1794.

Bonomi, F., Milito, R., Zhu, J. and Addepalli, S., 2012, August. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing* (pp. 13-16).

Buyya, R. and Srirama, S.N. eds., 2019. *Fog and edge computing: principles and paradigms*. John Wiley & Sons.

Carnevale, L., Celesti, A., Galletta, A., Dustdar, S. and Villari, M., 2018, May. From the cloud to edge and IoT: a smart orchestration architecture for enabling osmotic computing. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)* (pp. 419-424). IEEE.

Giust, F., Verin, G., Antevski, K., Chou, J., Fang, Y., Featherstone, W., Fontes, F., Frydman, D., Li, A., Manzalini, A. and Purkayastha, D., 2018. MEC deployments in 4G and evolution towards 5G. *ETSI White paper*, *24*(2018), pp.1-24.

Gupta, H., Nath, S. B., Chakraborty, S., & Ghosh, S. K. (2016). SDFog: A software defined computing architecture for QoS aware service orchestration over edge devices. In *arXiv (cs.NI)*. http://arxiv.org/abs/1609.01190

Habak, K., Ammar, M., Harras, K.A. and Zegura, E., 2015, June. Femto clouds: Leveraging mobile devices to provide cloud service at the edge. In *2015 IEEE 8th international conference on cloud computing* (pp. 9-16). IEEE.

Hamdan, S., Ayyash, M. and Almajali, S., 2020. Edge-computing architectures for internet of things applications: A survey. *Sensors*, *20*(22), p.6441.

Khan, A.A., Abolhasan, M. and Ni, W., 2018, January. 5G next generation VANETs using SDN and fog computing framework. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (pp. 1-6). IEEE.

Kreutz, D., Ramos, F.M., Verissimo, P.E., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S., 2014. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, *103*(1), pp.14-76.

Okay, F.Y. and Ozdemir, S., 2018. Routing in fog-enabled IoT platforms: A survey and an SDN-based solution. *IEEE Internet of Things Journal*, *5*(6), pp.4871-4889.

Salaht, F.A., Desprez, F. and Lebre, A., 2020. An overview of service placement problem in fog and edge computing. *ACM Computing Surveys (CSUR)*, *53*(3), pp.1-35.

Soua, A. and Tohme, S., 2018, February. Multi-level SDN with vehicles as fog computing infrastructures: A new integrated architecture for 5G-VANETs. In *2018 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)* (pp. 1-8). IEEE.

Sun, X. and Ansari, N., 2016. EdgeIoT: Mobile edge computing for the Internet of Things. *IEEE Communications Magazine*, *54*(12), pp.22-29.

Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S. and Sabella, D., 2017. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, *19*(3), pp.1657-1681.

Toor, Y., Muhlethaler, P., Laouiti, A. and De La Fortelle, A., 2008. Vehicle ad hoc networks: Applications and related technical issues. *IEEE communications surveys & tutorials*, *10*(3), pp.74-88.

Truong, N.B., Lee, G.M. and Ghamri-Doudane, Y., 2015, May. Software defined networking-based vehicular adhoc network with fog computing. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)* (pp. 1202-1207). Ieee.
Verma, M., Bhardwaj, N. and Yadav, A.K., 2016. Real time efficient scheduling algorithm for load balancing in fog computing environment. *Int. J. Inf. Technol. Comput. Sci*, *8*(4), pp.1-10.

Verma, A., "Supervised Edge detection using serverless computing," Research in computing, National college of Ireland, September 2022.

Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J. and Jue, J.P., 2019. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, *98*, pp.289-330.