Architectural Design and Evaluation for Distribution and Governance of Vehicular Information in Smart Cities

Roger Young

A Thesis Submitted for the Degree of Doctor of Philosophy

Supervisors:

Dr. Sheila Fallon

Dr. Paul Jacob

Funded By

Irish Research Council: EBPPG/2018/206

Tech-Auto Ltd

Submitted to Athlone Institute of Technology, 20th July 2020

Connect & Discover

DECLARATION

I hereby declare that the work presented in this transfer thesis is solely my own work and that to the best of my knowledge the work I present is original except where indicated by a reference to other respective authors or organisations.

Signed: ______________________________

Roger Young

Date:

Acknowledgements

The journey of a PhD was not just four years, but eight. To say it’s a long and difficult journey is an understatement. However, it is also an enjoyable journey. One that I could not have taken alone. For this reason, there are many people I would like to sincerely thank. First, to the amazing staff at AIT. From day one I felt at home and it was a pleasure to learn from such a great group of peers. I wish to express my sincere gratitude to my main supervisor Dr. Sheila Fallon, Lecturer in the Dept. of Software and Electronics Engineering in Athlone Institute of Technology, for providing me with the opportunity to complete my PhD. I could not have asked for a better supervisor and I am so grateful for your knowledge, time, feedback and enthusiasm over the past 4 years. I would also like to thank my second supervisor Dr. Paul Jacob. Your feedback and experience has been invaluable throughout the course of this work. And to my fellow Post Graduate researchers who have been there on many occasions to provide valuable insight and support when needed, I thank you.

This work was funded by the Irish Research Council and Tech-Auto Ltd for which I am very grateful, with a special thank you to Dr. Denis O’Dwyer for this amazing opportunity.

Finally, I owe everything to my beautiful wife Una and three fantastic children. This has been a long road for all of us, and without your continuous love and support, these past eight years would not have been possible.

Abstract

The rate at which we generate data nowadays has instigated a migration of data processing concepts and technologies from the Cloud-Computing paradigm. The potential destructive impact of "Internet of Things" on the current network infrastructure has been addressed by the rise of Edge Processing and Fog-Computing. However, it is clear that imminent technologies such as Smart Cities and Connected Vehicles need a collaborative platform of Edge/Fog/Cloud for success. Aggregated vehicle data can provide "data blanketing" of every street throughout a city, giving an accurate snapshot of current traffic and driving behaviour. Gathering such data is referred to as Vehicle Telematics, and can be obtained through On-Board Units (OBU). Modern vehicles generate up to 5GB an hour, with newer models generating far more. Processing and transmitting this information in the most efficient way possible is a hotly researched topic.

In this thesis we design, implement, and evaluate a novel data-centric architecture with capabilities of performing bi-directional communication between Edge/Fog/Cloud nodes. Our architecture, WAVE-Flow, consists of a combination of Flow Based Programming and the WAVE communication protocol, with the focus on Vehicle-to-Infrastructure scenarios (V2I). With much of the current literature focusing on the communication protocols between OBUs and Road-Side Units (RSU), this work develops a set of mechanisms within the nodes that enhances governance over data processing and distribution.

WAVE employs the IPv6 protocol, and also introduces a WAVE Short Message Protocol (WSMP). Due to high mobility, connectivity between vehicles and RSUs may be unstable. Furthermore, RSUs may only be situated at hotspots throughout a city, such as traffic intersections. This may be problematic as vehicles may be out of range before data transmission is finished. To combat this issue, we develop an in-vehicle mechanism called W-V6 that switches between WSMP and IPv6 depending on its proximity of an RSU.

We have extended the WAVE protocol with a specific message set designed for vehicle telematics. The WAVE-Flow architecture stores the message locally in-vehicle until requested by an RSU. However, if, for example the vehicle moves out of the transmission range of the RSU, and can no longer transmit via WSMP, the message will be packaged as a UDP packet and sent via IPv6 to the RSU. Results will show our W-V6 mechanism extends the service area of the RSU by hybridizing the communication protocols available in multi-interface vehicles.

Contents

List of Figures

List of Tables

Abbreviations

AC
  Access Category, 33
AMS
  Advanced Message Set, 112
AODV
  Ad hoc On-Demand Distance Vector, 39
AVL
  Automatic Vehicle Location, 83

BSM
  Basic Safety Message, 36

CAM
  Cooperative Awareness Message, 36
CAN
  Controller Area Network, 23
CSMA/CA
  Carrier Sense Multiple Access with Collision Avoidance, 33
CW
  Contention Window, 34

DAGLADS
  Distribution And Governance of LArge DataSets, 3
DSR
  Dynamic Source Routing, 39
DSRC
  Dedicated Short Range Communication, 29

ECU
  Electronic Control Unit, 23
EDCA
  ; Enhanced Distributed Channel Access, 33

FBP
  Flow Based Programming, 41
FMS
  Fleet Management System, 25

HDF
  Hortonworks Data Flow, 11

OBDII
  On-Board Diagnostics II, 24
OBU
  ON-Board Unit, 4
OCB
  outside the Context of a BSS, 30

PLR
  Packet Loss Ratio, 40
PSC
  Provider Service Context, 33
PSID
  Provider Service Identifier, 32

QoS
  Quality of Service, 33

RTS/CTS
  Request to Send / Clear to Send, 33

UDP
  User Datagram Protocol, 4

V2I
  Vehicle to Infrastructure, 2
V2V
  Vehicle to Vehicle, 2
VANET
  Vejicle Ad-hoc Network, 29

WME
  Wave Management Entity, 31
WRA
  Wave Routing Advertisement, 33
WSA
  Wave Service Announcements, 31
WSM
  Wave Short Messages, 4
WSMP
  Wave Short Message Protocol, 4
WSN
  Wireless Sensor Network, 8

1 Introduction

1.1 Context and Motivation

Over the next few years we are going to witness numerous unifications and crossovers of some of the more well-known buzzwords/technologies of recent times. Internet of Things, Big Data, Real-time Processing, Edge and Fog-Computing, Connected Vehicles, and so on, will no longer be viewed as specific threads of research. Instead, the focus will turn to how much these technologies are compatible with one another and can enhance each other's success stories. One prime example is Smart Cities. Smart Cities are cities that embrace the IoT concept to improve quality of life for its dwellers. Gathering data from heterogeneous devices throughout the city, in-turn, generating big data that produces valuable trends and insight for city developers. Data is commonly regarded as the newest natural resource, a resource which in this scenario can vastly improve mobility, public utilities, safety and a cleaner environment.

Since the introduction of Cloud-Computing, we have seen a massive surge in data related technologies. The Cloud allowed companies to invest in more sensor devices, which in turn provided more valuable information that could be translated to profit. Managing large amounts of data was made possible by technologies such as Hadoop and MapReduce. However, newer applications required faster response times, meaning traditional batch processing techniques were no longer feasible. Real-Time processing technologies such as Apache Spark and Apache Storm addressed this issue. These platforms opened up new doors for developers to apply algorithms and logic to data instantaneously which suited many scenarios, the automotive industry in particular.

Year on year we are generating an exponential amount of data, and although real-time processing helps manage this huge quantity, it is not enough. This brought about the introduction of Edge-Computing, where devices that generate the data are given some processing capability to reduce transmission to the Cloud. This was followed by the introduction of Fog-Computing, where existing network nodes are given processing capability between the Edge and Cloud. Fog-Computing bridges the gap between the Cloud and end devices by enabling computing, storage, networking, and data management on network nodes along the IoT-to-Cloud path. A combination of Edge/Fog-Computing, Real-Time Processing and the Cloud makes the Internet of Things possible. Otherwise, sending all data to the Cloud would just result in network overload, latency issues, and overall poor quality of service.

Connected Vehicles within a Smart City are set to play a major role. Simply put, vehicles are computers on wheels, with over 2500 sensors that provide highly accurate information on traffic and driving behaviour, weather and road conditions, to name but a few. Recent advancements in communication protocols between vehicles (V2V), and vehicles and surrounding infrastructure (V2I) have enhanced safety issues greatly in urban areas, but also opened opportunities for non-safety applications. One such example is vehicle telematics/fleet management. Recently, it is becoming more apparent, that vehicle telematics may hold tremendous potential as a key enabler for the development of Smart Cities. Rich vehicle data has potential to make, not just Smart Cities, but our environment, a safer and cleaner place.

1.2 Problem Statement & Contributions

Is it possible to design a data-centric architecture with the necessary performance to support bi-directional communication between Edge/Fog/Cloud nodes in an IoT paradigm?

With projected figures of 50 billion connected devices by 2020 [1] , how data is distributed and governed will play a significant role in the success of the oncoming Internet of Things paradigm. The current network infrastructure is already strained, forcing many of the world's leading telecommunication companies to invest in Research and Development for potential solutions in dealing with the ever-rising growth of data. This brought about the introduction of Edge-Computing, and later Fog-Computing. However, currently there are a lack of IoT platforms for developers to implement their applications on Edge devices. This is also one of the key gaps in realizing Fogs potential, along with applicable use cases. Therefore, in defining a reference architecture for IoT applications, certain requirements must be addressed, such as evaluating a suitable programming platform, local processing potential, data capturing techniques, coordination and governance capability. With this in mind, the following research questions must be addressed.

1) *In terms of communication and collaboration between nodes, to what degree can Flow Based Programming address the heterogeneity of services and applications in Vehicle-to-Infrastructure scenarios (V2I)?*

The number of applications utilizing Flow Based Programming (FBP) has been growing over the past number of years, in line with the introduction of IoT. For this reason we aim to design, implement and evaluate a FBP inspired architecture called DAGLADs (Distribution & Governance of Large Datasets). The aim of DAGLADs is to enhance Edge Processing, with the added benefit of instant governance from the Cloud. We evaluate DAGLADS against the current state of the art Cloud approach, with the aim of maintaining computational accuracy while minimizing transmission to the Cloud. Other important aspects for a successful IoT platform such as advanced Edge analytics capabilities and collaborative task offloading to the Cloud will also be taken into consideration during the evaluation. DAGLADS will be presented as an Edge/Cloud platform. However, our overall goal is to create an architecture that spans across Edge/Fog/Cloud. For this, we need a suitable use-case. Interaction between Connected Vehicles and Smart Cities infrastructure is a prime example of Edge/Fog/Cloud, therefore, if one was to focus on a platform for IoT applications, Connected Vehicles would be an optimal use case, which leads to addressing the following questions.

2) *For vehicle telematics to be considered a key enabler in the development of Smart Cities, which in-vehicle data capturing technology provides the most accurate sensor data?*

Connected Vehicles provide a wealth of data that may be used in the development of Smart Cities. Although future OBUs may ingest data directly from the vehicle network, currently most industry based technologies and also current literature capture data via the OBDII standard. This is due to its ease of access and installation. OBDII is often used as ground truth when comparing to other fleet management systems such as GPS based technologies. However, another less utilized standard exists called FMS. Originally only for buses and trucks, FMS can now be installed in passenger vehicles via a FMS gateway. As our focus is on vehicle information in urban areas, choosing the most accurate data capturing technology is imperative. For this reason we evaluate and compare both standards simultaneously, providing analysis on the most reliable standard to use when monitoring driving behaviour, traffic behaviour, and environmental factors such as CO_2 emissions.

3) *Can a Multi-Tier Flow Based Programming Architecture enhance the distribution and governance of vehicle sensor information in Smart Cities?*

The initial design of our reference architecture was to perform intelligent distributed data processing within an IoT context. However, with Connected Vehicles interaction with surrounding infrastructure considered a growing research topic, we redesign our DAGLADs architecture to compliment the inclusion of RSUs (Fog Devices) and the de-facto V2V communication protocol WAVE. One of the novelties of WAVE is that higher layer applications have the ability to determine lower layer parameters such as data rate, transmission power etc. We aim to enhance this concept with a novel architecture called WAVE-Flow in which higher layer applications in the RSU can modify lower layer parameters on vehicle OBUs. In essence, RSUs have governance over the transmission of information from the vehicles. This will be achieved by bringing together the concepts of FBP and WAVE with the aim of shedding new light on the role FBP may play in V2V/V2I scenarios.

WAVE supports two messaging protocols, IPv6 and WSMP. WSMP is a fast, one-hop messaging system for dissemination of safety beacons, whereas IPv6 can be utilized for internet connectivity and multi-hop purposes. In terms of a unicasting scenario such as vehicle telematics in V2I, WSMP provides higher throughput with lower packet loss. However, a major drawback is once out of range, it can no longer transmit to the RSU. To combat this issue, we design and evaluate an in-vehicle mechanism called W-V6 that switches from WSMP to IPv6 once the vehicle is out of proximity of an RSU.

As a use case for WAVE-Flow, we introduce a city wide fleet management platform. We also introduce a specific message set for vehicle telematics. As part of the use case, exhausting simulations of numerous combinations of lower layer parameter modifications will be tested. To the best of the authors' knowledge, this will be the first body of work that takes into consideration data rate, QoS mechanisms, and data granularity for the enhancement of packet delivery in a V2I unicast scenario. It is also the first body of work to perform an in-depth comparison of WSM and UDP in one-hop scenarios. In our use-case, RSUs act as control portals to the vehicles in its vicinity. Mechanisms on the RSU can determine at what granularity and data rate vehicles transmit their messages, dependant on external parameters such as current traffic congestion. WAVE-Flow can be considered as the main contribution of this thesis.

1.3 Publications

The idea and the contributions presented in this thesis are part of several peer reviewed research papers. In this subsection we give the list of our publications sorted by date. Figure 1 presents a hierarchical view of our reference architecture. As shown, our architecture is evaluated in two platforms, DAGLADS and WAVE-Flow. The bottom layer shows the different areas in which we contributed with a numbered link to our publications.

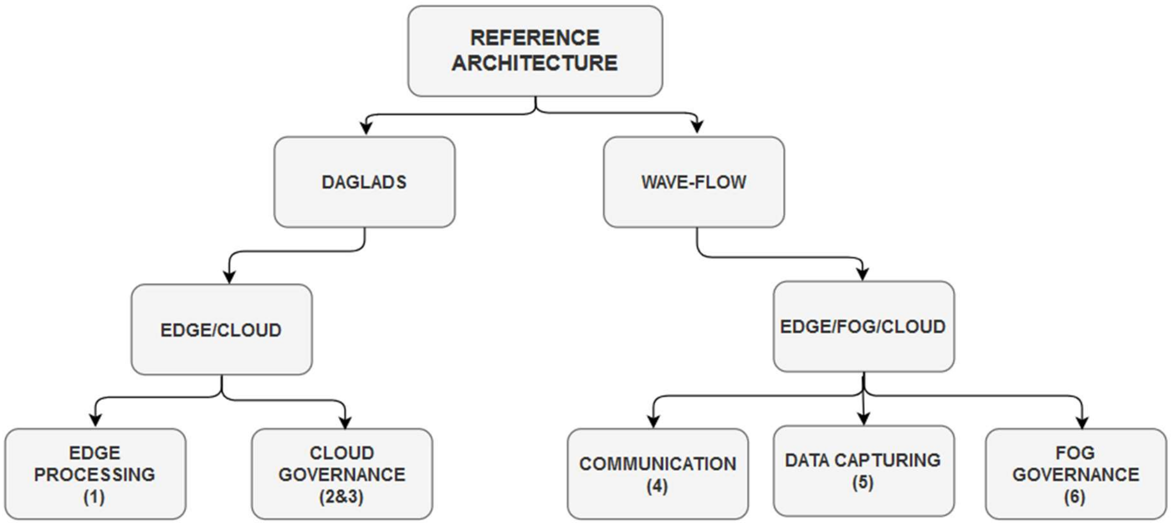


Figure 1: Hierarchical Structure of Reference Architecture & Contributions

1) **An Architecture for Intelligent Data Processing on Edge Devices in an IoT Environment**

Young, R., Fallon, S., & Jacob, P. (2017, April). An architecture for intelligent data processing on iot Edge devices. In *2017 UKSim-AMSS 19th International Conference on Computer Modelling & Simulation (UKSim)* (pp. 227-232). IEEE.

2) **A Governance Architecture for Self-Adaption & Control in IoT Applications**

Young, R., Fallon, S., & Jacob, P. (2018, April). A Governance Architecture for Self-Adaption & Control in IoT Applications. In *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)* (pp. 241-246). IEEE

3) **Dynamic Collaboration of Centralized & Edge Processing for Coordinated Data Management in an IoT Paradigm**

Young, R., Fallon, S., & Jacob, P. (2018, May). Dynamic collaboration of centralized & Edge processing for coordinated data management in an IoT paradigm. In *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)* (pp. 694-701). IEEE.

4) **A Flow Based Architecture for Efficient Distribution of Vehicular Information in Smart Cities**

Young, R., Fallon, S., Jacob, P., & Dwyer, D. O. (2019, October). A Flow Based Architecture for Efficient Distribution of Vehicular Information in Smart Cities. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)* (pp. 93-98). IEEE.

5) **Vehicle Telematics and its Role as a Key Enabler in the Development of Smart Cities**

Young, R., Fallon, S., Jacob, P., & Dwyer, D. O. (2020). Vehicle Telematics and its Role as a Key Enabler in the Development of Smart Cities. *IEEE Sensors Journal.*

6) **A Flow Based Architecture for a Multi-User Fleet Management System in Smart Cities**

To be submitted to IET Smart Cities

1.4 Thesis Layout

Chapter 2 provides an in-depth review on a number of relevant areas to our thesis, including Edge/Fog Computing, Smart Cities, Connected Vehicles, WAVE, and Flow Based Programming. In chapter 3 we design an IoT architecture with strong focus on real-time Edge processing and bi-directional communication. This chapter presents the design and methodology of our DAGLADS architecture, while detailing the fundamental requirements, technology used, architectural entities and the system components within each entity. Chapter 3 concludes with a series of implementations and results of DAGLADS.

As our focus turns to Connected Vehicles within a Smart City, chapter 4 discusses the vehicle network, and analyses in-vehicle data capturing techniques. The goal of chapter 4 is to provide insight on the most accurate data capturing standard in terms of monitoring driving and traffic behaviour. In chapter 5 we introduce our multi-tier architecture WAVE-Flow. The WAVE-Flow architecture builds on the methodology and knowledge gained from chapter 3 and 4, allowing us to incorporate our reference architecture in a V2I use case. Finally, chapter 6 concludes our overall findings and limitations of the thesis.

2 Literature Review

With considerable predictions of connectivity to the web over the coming years, ongoing research exists to address the issues surrounding the exponential increase of data. The introduction of Edge-Computing, where Edge devices come with the capability to process and analyse newly generated data, now allows for the distribution of some data analytics from the Cloud to the Edge devices. Although Edge and Cloud-Computing are evolving rapidly, architectures that efficiently coordinate computations in a dynamic manner are still in their infancy. While Edge-Computing may perform better in low latency real time prediction, the Cloud approach is advantageous in storage and processing power. It is evident a collaborative system that avails of the benefits from both models is fundamental to the success of IoT [2].

The more recent introduction of Fog-Computing can be seen as computation taking place in the layer between the Edge and the Cloud. Currently, there are many definitions of Fog-Computing, with much literature still interchanging between Fog and Edge. For example, [3] provides an in-depth survey on Edge-Computing and its role as a key enabler for many future technologies like 5G, IoT, augmented reality and vehicle-to-vehicle communications by connecting Cloud-Computing facilities and services to the end users. However, the authors categorized and classified the state of the art in Edge-Computing as Cloudlets, Fog and Mobile Edge-Computing. In contradiction to this, the authors of [4] provide insight into Fog-Computing and similar programming models, stating Fog-Computing provides computing, networking, storage, and control anywhere from cloud to things; while Edge-Computing tends to be limited to computing at the edge. In [5], it states that despite the existence of the concept for several years now, commercial deployments of Fog-Computing are yet to take off. Part of the reason is there are still an inadequate number of widely deployed or critical applications that find Fog beneficial. There also seems to be a lack of clarity on the application model, runtime and management environments for a Fog platform.

2.1 Migration of Computation from the Cloud

Traditionally, IoT devices, and wireless sensor networks (WSNs) were commonly designed to transfer data to remote servers and computing Clouds as discussed in [6], [7], and [8]. Cloud offers infrastructure, platform, and software as services (IaaS, PaaS, SaaS). Application developers can use a variety of these services depending on the needs of the applications they develop. Recent work such as [9] propose a real-time job scheduler in Hadoop for Big Data. The scheduler aims to manage cluster resources in such a way that the real time jobs will not be affected by the long running (batch jobs), and vice-versa. The case study is applied as support for Smart City applications, taxicabs in particular. Although efficient in its design, all data is transferred to a single location in a completely centralized scheduler.

The authors of [10] propose an architecture for streaming spatio-temporal event processing, analysis and near real-time visualization. It is comprised of fully open source software and focuses on a use case involving a fleet of snowploughs. Information on the plows are published to the public, as well as road coverage data. Technologies such as LocationTech GeoMesa, GeoTools, and GeoServer are used to enable geospatially-aware complex event processing (CEP) solutions. There are some challenges when it comes to processing stream geospatial events, such as handling differences in event and processing time. However, the inclusion of software such as Apache Storm [11] and Apache Kafka [9] into this architecture addressed such problems. The authors of [12] propose a Big Data pre-processing quality framework that focuses on the Quality of Big Data (QBD). This framework aims at solving data quality issues that occur when attempting to apply data quality concepts to large data sets. All of the above-mentioned papers are versions of a centralized approach, which leaves them vulnerable to latency issues and network load. Both [13] and [14] reiterate the well-known fact that sending all data to the Cloud will require prohibitively high bandwidth if it is to provide greater service and faster quality. Also, in terms of mission critical data such as safety information for Connected Vehicles, the Cloud paradigm alone, is obsolete. Figure 2 shows a typical Cloud-centric scenario in which all sensor data is sent to the Cloud for analytics and storage. As shown in the image, this leaves potential for network bottlenecks when the bandwidth cannot sustain the influx of data to the Cloud.

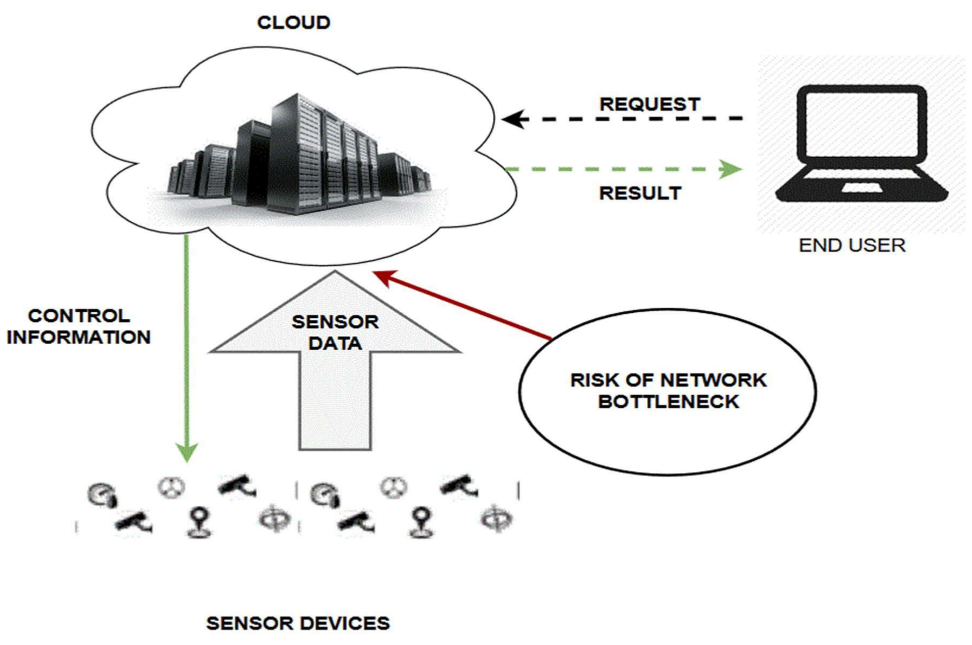


Figure 2: Typical Cloud-Centric Scenario

2.1.1 Edge-Computing

Over the past three to four years, we have noticed a shift in the data-mining paradigm. There has been a significant rise in published work in this timeframe that focuses on Edge-Computing. It may also be a fair assumption to say that over the past 24 months in particular, Fog-Computing is the computing model receiving most attention. The rest of this section will begin with reviewing work solely based on Edge analytics before moving onto recent Fog-Computing literature.

In Edge-Computing, physical assets like pumps, generators and motors are physically wired into a system utilizing a programmable automation controller (PAC). PACS add intelligence that allows for Edge analytics close to the sensor devices. As discussed, Edge-Computing has drawn much attention in both industry and academia. This is due to the widely accepted fact that processing close to source is paramount to the success of IoT. Connected Vehicles provide a perfect example for such a scenario. With each vehicle generating up to 5GB of data an hour, and future vehicles predicted to generate far more, it is inconceivable to send this data to the Cloud, especially in life and death situations such as autonomous driving. Mission critical decisions such as these require on board decision-making, eliminating the risk of network failure.

There is much literature in this area that presents solutions to reduce data transmission to the Cloud. The work presented in [15] examines the benefits of data mining on the wireless, battery-powered, smart sensing devices at the Edge points of IoT. The authors implement three specific algorithms: Linear Spanish Inquisition Protocol (L-SIP), ClassAct, and Bare Necessities (BN). These algorithms fall under GSIP, or General Spanish Inquisition Protocol (SIP). Under SIP, nodes only send unexpected information. The goal of this work was to transform data at source into valuable information, in turn reducing packet transmissions, energy use, and storage space. Results showed packet reduction of between 95% - 99.98% demonstrating the importance of Edge mining in an IoT environment.

Nectar Agent is presented in [16], a solution that automates the switching between different data handling algorithms on Edge devices. It also has the capacity of adjusting and analysing data reduction methods. The authors introduce three flavours of a new algorithm capable of performing real-time reduction of incoming time series items based on the concept of Perceptually Important Points. The paper present a method called Streamification, where data reduction algorithms work upon data streams. Evaluations are made using real datasets from street, household, and robot sensors. Accuracies of 76.1 % to 93.8 % are reached.

As demonstrated by the papers above, several interesting proposals have emerged recently from the realization that not all IoT generated data are of equal importance. This has led to certain strands of research to focus on sending “important” data to the Cloud. However, although this may be sufficient in certain scenarios, many companies may not want to simply discard raw data as it may still obtain valuable information. In many cases, it may prove beneficial to focus on filtering and transmitting “relevant” data at source, but also have the potential to temporarily locally store “all” data for batch processing at a later date.

Hortonworks [17] demonstrated the simulation of bi-directional data communication between an on-vehicle platform and the Cloud. This was achieved by loading Apache MiNifi onto a custom Qualcomm modem located in a connected car, allowing the vehicles to transmit data to their HDF (Hortonworks Data Flow) platform [18]. The demo highlighted how to deliver critical capabilities for vehicle communication. The centralized HDF platform could process key data such as speed and geo-location in real time. MiNifi could manage how and when to transmit much larger but less time-relevant data, (system diagnostics, etc.) This data could be batched on the vehicle and sent in bursts over known Wi-Fi locations. This is an effective solution as bandwidth over LTE is expensive. When testing our proposed architecture DAGLADS, HDF will be used as the centralized approach for comparison. This is because HDF can be viewed as current state-of-the-art in real time processing.

FocusStack [19] is an Edge/Cloud platform created by combining OpenStack, one of the most popular open source cloud management platforms, with the AT&T Labs Geocast System (ALGS), a multi-tiered geographic addressing (GA) network subsystem that allows packets to be sent to (all devices in) a geographic region instead of a specific set of IP endpoints, as in IP unicast or multicast. Linux based containers are deployed to the edge devices which in turn allows developers to implement custom applications. FocusStack is similar to our proposed Edge/Cloud platform, with the main difference being our platform is not confined to deployment of applications to a specific region.

Similar Edge development platforms include NodeRED and Apache Edgent. However, NodeRED does not have a general way for configuring applications dynamically, and Edgent does not provide any "deployment" mechanisms, it does however, recommend to FTP the application to the device and modify the device to start the application upon start-up [20]. The work in [21] presented the foundations of a new ongoing Horizon 2020 research project called LightKone that extends Edge-Computing for general-purpose computation, making it more scalable and flexible, and providing a set of mechanisms to simplify development and deployment of applications. The LightKone approach is based on the use of synchronisation-free shared mutable data combined with robust and efficient hybrid gossip communication primitives.

Although there is a shift towards Edge-Computing, it is not without its own shortfalls. The authors of [22] discussed the issues that surround Edge processing. It points out that Edge devices are most often battery powered. This is problematic as sending and receiving data drains energy quickly on mobile devices and smart sensors. Furthermore, [23] stated that although there are ongoing parallel advances in Cloud-Computing and Edge-Computing, interactions between these platforms in handling live data analytics from the communication perspective have been rarely investigated in the literature. This statement is reiterated in [24]. Taking all of this information into account, it further cements our research question in the need for a collaborative platform for IoT programming. Figure 3 shows a typical Edge-Computing paradigm. Analytics performed closer to where the data is generated reduces traffic to the Cloud, which, in turn, reduces the risk of network latency. This solution is also cost effective for businesses in terms of bandwidth and storage costs.

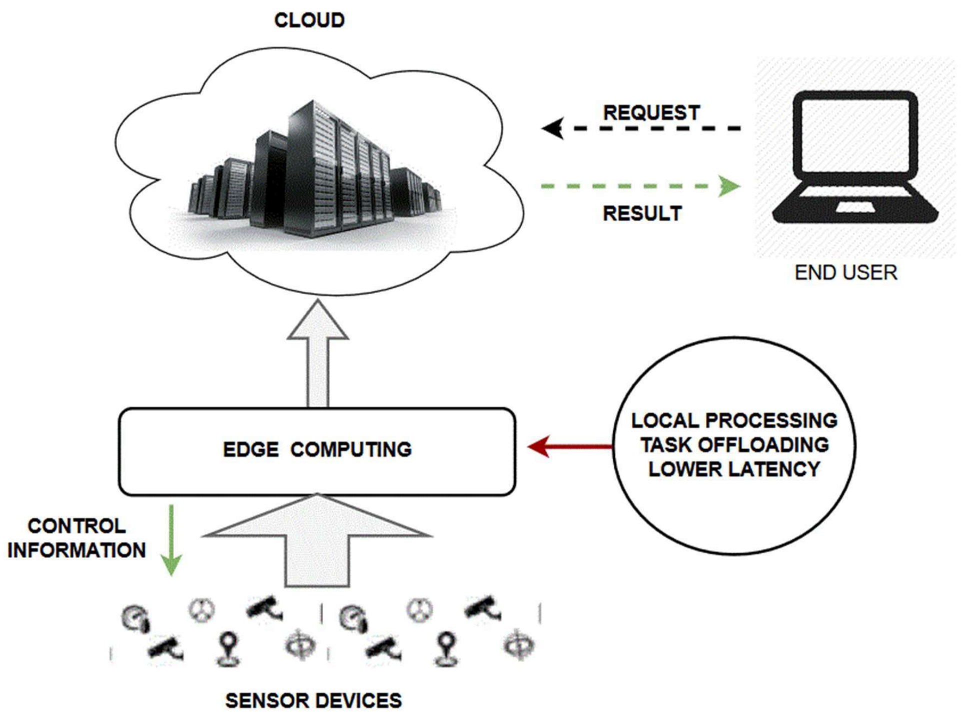


Figure 3: Typical Edge-Computing scenario

2.1.2 Fog-Computing

As previously mentioned, Fog-Computing has received much interest recently. As stated in [25], Fog is an architecture that distributes computation, communication, control and storage closer to the end users along the Cloud-to-things continuum, including numerous types of devices. Early works such as [26] and [27] introduced Fog concepts to aid in analytics of sensor data. The authors of [26] proposed IFoT (Information Flow of Things) middleware. IFoT provides four functions: 1) the distribution of tasks issued by application software into subtasks and distributed execution of the sub-tasks over multiple Edge and Fog devices. 2) Distribution of data streams over Fog devices. 3) Real-time analysis of the data streams, and 4) Seamless integration of sensors and actuators. A use case involving a face recognition system for person tracking was provided using a Raspberry Pi by locally processing video streams in real-time and a distributed manner.

Krikkit [28] is an open-source solution initiated by Cisco, but has been acquired by Eclipse. It is a publish/subscribe mechanism where rules are registered on the Edge/Fog gateways that communicate with sensors. It is in the process of specifying a data format and a mechanism for "telling the network-Edge devices" which data to forward and how. Krikkit is a promising concept but is still in development stage; also, it is uncertain as to whether it will be implemented only on router like devices. SpanEdge [29] is a two-tier processing platform that utilizes close to the edge devices such as routers and gateways. The approach of SpanEdge is to distribute stream processing applications across central and near-the-edge data centres in order to reduce the response latency and bandwidth consumption and thus improving the performance of stream processing applications. Although the name may suggest Edge processing, it is evident in this work they are referring to a Fog/Cloud processing platform. In fact, the author stated SpanEdge leverages the state-of-the-art solutions for hosting applications close to the network edge, such as cloudlets and Fog.

Authors of [30] described Fog-Computing as a model to complement the Cloud through the distribution of the computing-plus-networking resources from remote data centres towards Edge devices. The goal is to save energy and bandwidth, while simultaneously increase the QoS level provided to the users. As a consequence, Fog Nodes (FNs) are virtualized networked data centres which run atop access points at the Edge of the access network, in order to give rise to a three-tier IoE/Fog/Cloud hierarchical architecture. This is in line with our proposed architecture, in which RSUs can be seen as the Fog nodes in a V2I scenario. This work also details a high level overview of a Fog architecture in a

Smart City, citing 4 layers; the physical layer where data is generated, the network layer guarantees data transport between the Edge devices and the overall Smart City infrastructure, the fog layer that performs real time aggregation and orchestration and finally the supervision layer that provides user friendly interfaces offered by the Smart City ecosystem. This is similar to our city wide architecture, however, in our use case, the fog layer and the network layer will be intertwined, as RSUs act as network devices with processing capabilities.

As discussed in our problem statement, and one of the underlying objectives of our work, [31] reiterated the fact that there is still a major challenge for developers to program their services to leverage the benefits of Edge and Fog-Computing. The authors presented FogFlow, an IoT Smart City platform. To showcase their platform they presented three use cases, unfortunately none were vehicle related for comparison. In their architecture, three entities are described; Cloud, Edge nodes and devices. Devices in this work refer to nodes such as vehicles, and cite that devices may include computation and communication capabilities. However, in FogFlow, the work was focused on the Fog nodes and Cloud doing the processing, and there is no bi-directional collaboration between Edge/Fog/Cloud, which will be a necessity in many scenarios.

Figure 4 shows a typical Fog-Computing paradigm. As shown, Fog-Computing aims at processing on network devices between the Edge and Cloud. Matt Vasey, director of IoT business development, Microsoft and an Open Fog Consortium officer, stated that “Edge is to Fog as apple is to fruit”. Therefore, it is perceivable that both Edge and Fog-Computing will work in unison to reduce data transmission to the Cloud.

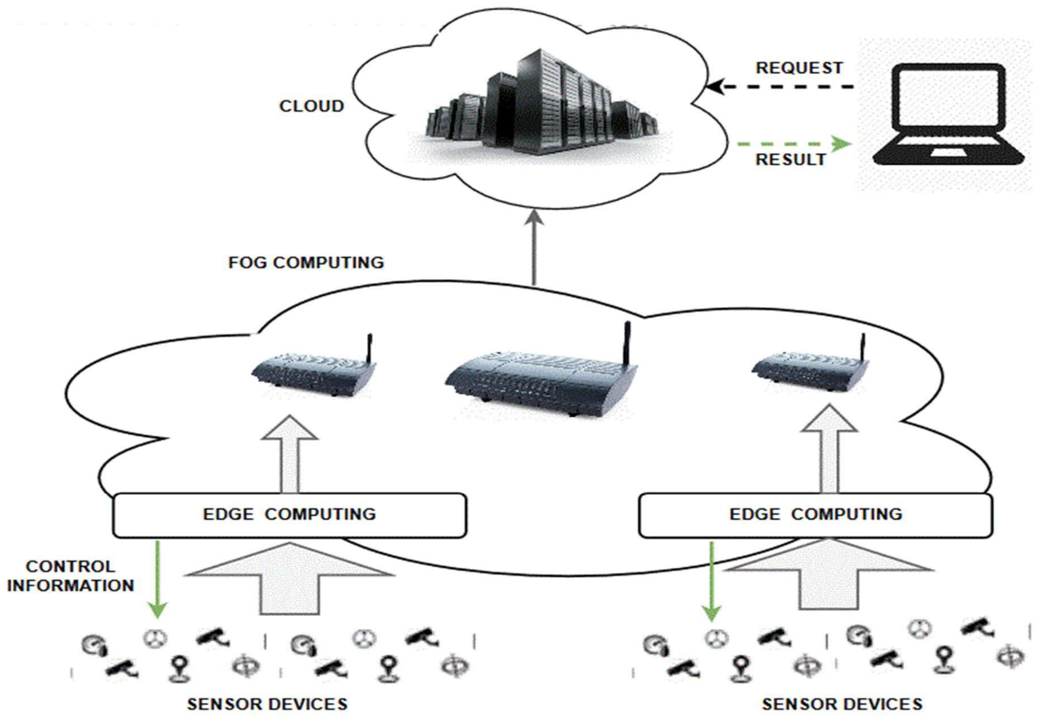


Figure 4: Typical Fog-Computing scenario

2.1.3 Paradigm Shift Review

It is clear that the introduction of Edge/Fog will play a major role in future technologies. It is also clear that Edge /Fog are not here to replace Cloud, but to complement it. This is further stated in [32], which reviewed the state-of-art of the analytics network methodologies, which are suitable for real-time IoT analytics. This work provides a thorough study of the existing IoT platforms, while stressing the importance of real-time analytics. Furthermore, [5] reviewed various dimensions of system architecture, application characteristics and platform abstractions that are manifested in this Edge, Fog and Cloud ecosystem. The author stated that the current lack of a platform ecosystem to design and run IoT applications that can use the Fog paradigm, is one of the key gaps in realizing Fogs potential. The authors also state that although IoT Edge management technologies such as Azure IoT Gateway and Amazon AWS Greengrass are becoming available, application platforms are still in their infancy.

2.2 Fog in Smart Cities

Throughout the world, major cities are experiencing ever-increasing urbanization and mobility challenges. Current estimates of 55% of the world's population residing in cities and towns are set to increase to 68% by 2050. An ever-growing population is adding much pressure on existing infrastructure, forcing city leaders to look towards technologies such as "Internet of Things" to aid in improving municipal quality of life, transforming cities into "Smart Cities". The introduction of IoT laid the foundations of Smart Cities, making real the vision of complete connectivity and interaction amongst heterogeneous devices. In academia, the definition of a Smart City is "an urban-development vision to integrate information and communication technology (ICT) and IoT technology in a secure fashion to manage a city's assets." However, a more favourable explanation was coined by R. Giffinger and H. Gudrun when quoting "A city becomes smart only if the performance indicators improve for several interconnected areas, such as the economy, people, governance, the environment, living, and mobility".

Smart Cities are only possible with a combination of Edge/Fog/Cloud. With millions of heterogeneous devices generating large quantities of data, it is simply infeasible to send everything to the Cloud. There are numerous Smart City architectures in the literature which were reviewed to provide us some insight into current state of the art. The vision of SMARTIE (Secure and sMARter ciTIEs data management) [33] and [34] is to create a distributed framework for loT-based applications storing, sharing and processing large volumes of heterogeneous information. This is an EU funded project, that although primarily focuses on privacy and security, may be implemented in many scenarios. However, the overall structure of SMARTIE is more centralized than our proposed architecture, providing specific APIs for different applications throughout the city.

A Fog based architecture for Smart Cities is proposed in [35]. In this article, the challenges and opportunities of applying Fog-Computing in a vehicle scenario are discussed, and a regional cooperative fog-computing-based intelligent vehicular network (CFC-IoV) architecture is proposed. CFC-IoV adopts a coordinator to provide low-delay coordination services for IoV applications. Similar to our proposal, CFC-IoV is an open system to heterogeneous networks. Authorized users, such as government service providers, and users, are able to add new entities to the Fog-Computing system. However, the bulk of the work in CFC-IoV is carried out by the distributed fog servers throughout the city, with the role of the RSUs limited to access points.

The Santander project [36] described the deployment and experimentation architecture of the Internet of Things experimentation facility being deployed at Santander city. It is an EU commissioned project that represents a unique city-scale experimental research facility. Their testbed structure is similar to our proposed architecture, in that it consists of a testbed server, a gateway tier and an IoT node tier. The Santander platform is adaptable to many use-cases such as parking management, environment monitoring, and participatory sensing where mobile phones are used as sensors. Because the experiment mostly uses Wireless Sensor Networks (WSNs) for ingesting data, the inter-tier connectivity is a mix of Wi-Fi, 3G, Bluetooth and Ethernet interfaces. Bonomi [37] of Cisco present a four tier architecture for Connected Vehicles in Smart Cities. The architecture proposed to consider the 4 steps that every IoV communication always involves which are: Embedded systems and sensors, Multi service Edge, Core, Data centre and Cloud.

DOCTraMS [38] is a system that monitors and disseminates traffic conditions using a decentralized infrastructure. On-board units and roadside units do not need to be interconnected or connected to a central computer. These elements exchange information to update their tables that describe the traffic conditions of each road segment presented. These characteristics allow the system to be used even in locations where there is no power infrastructure or cellular coverage.

The authors of [39] presented TrasoNET, an integrated network framework that provides real-time intelligent transportation services to Connected Vehicles by exploring the data analytics and networking techniques. TrasoNET focuses on providing vehicles a real-time status on local traffic. TrasoNETs consists of the following 4 components;

Access Infrastructure: The access infrastructure consists of evolved Node Bs (eNBs) and RSUs. Mobile devices like smartphones connect to eNBs via LTE, whereas vehicles connect to RSUs via DSRC or WI-FI.

Mobile Devices: Smartphones or vehicles.

Central controller: Connects to the eNBs and RSUs. It acts as an interface between the physical network routers and the network operators to specify network services.

Cloud: The majority of data analytics is performed here.

This work also introduced the concept of a regional RSU, a coordination node that connects to a number of local RSUs. Similar to [35], the role of RSUs seems to be little more than access points for vehicles. However, this is in contrast to the recent introduction of Cisco Kinetic [40]. Cisco's IoT platform Kinetic presents RSUs as powerful processing devices, similar to our proposal. In fact, Kinetic utilizes a flow based programming approach to perform decision making on data in motion in IoT scenarios. This is a focal point of our proposal. RSUs are set to have advanced processing capabilities, and as the closest infrastructure devices to the Edge, hold great potential in acting as governing nodes throughout an urban area. Such nodes can process and make decisions faster, gaining valuable insight from vehicle sensor information instantaneously.

It must be noted there are other works that utilize RSUs as coordination managers in vehicle applications. The authors of [41] utilized RSUs as coordination nodes to aid in formulating a distributed service model for optimizing the service placement on moving vehicles. In this work, a Control Node (CN), or cluster head, is chosen among a group vehicles that will update the cluster state in terms of location and resource availability to the RSU. The CN is chosen as it will be the node that has the greatest probability of staying in the cluster until the service has completed. Similarly, [42] proposed a system in which the RSU chooses a cluster head from a group of vehicles, in which the RSU and cluster head are responsible for the V2V communication with the aim of finding traffic congestion and the shortest path for data transmission between nodes.

2.3 Connected Vehicles & Data Capturing Techniques

Projections show that urbanization, combined with the overall growth of the world's population could add another 2.5 billion people to urban areas by 2050. [43]. This coincides with the increasing trend of our reliance on vehicles. The United States, claims an average of 812 cars for every 1,000 people. In China, the number of vehicles is expected to reach 250 million by the end of 2020 [39]. Connectivity is increasing around the world and its expansion to vehicles is no exception. With improvements in connectivity, sensing, and computation, the future will see vehicles used as development platforms capable of generating rich data, acting based on inference, and effecting great change in transportation, the human-vehicle dynamic, the environment, and the economy [44]. Increased vehicular sensing and connectivity creates troves of useful data. A growth in vehicle and Cloud-Computing power, as well as scalable data handling tools, has made gleaning critical insights from vast data sets tenable. This is further aided with the decreasing price and power consumption of microcontrollers, which are now integrated in many vehicle components. With this, local application development and data aggregation is now possible. There is a wealth of literature stating the importance of the role vehicles in urban areas may play [45]–[50], in terms of improving urban mobility, reducing pollution, and enhancing safety.

2.3.1 Vehicle Telematics

Vehicle telematics holds great potential in terms of making not just Smart Cities, but our environment, a safer and cleaner place. Initially, the goal of vehicle telematics was to help businesses lower operational costs, while monitoring driver behaviour and vehicle health. However, a growing number of companies are beginning to prioritize their fuel efficiency and carbon footprint. This change is not only driven by reducing fuel related costs, but also government initiatives to combat the rise in Greenhouse Gas Emissions (GHG). The transportation sector is responsible for a significant fraction of total Greenhouse Gas emissions, second only to energy production. Worryingly, while emissions from other sectors are generally decreasing, those from transportation have increased by 36% since 1990 [51]. This has led the automotive industry to invest in a volume of technologies that reduce the necessity of fossil fuels. However, widespread use of electric vehicles is still a distant vision, meaning other fuel saving technologies are required. One such method is Eco-Driving.

The characteristics of Eco-Driving are generally well defined and easily characterized [52]–[57]. They involve such things as accelerating moderately (moving up gears between 2000 and 2500 revolutions), reducing hard deceleration by anticipating traffic flow and upcoming signals, maintaining a steady driving speed, route choice, and eliminating excessive idling [57]. Compared to other solutions, the implementation of Eco-Driving is relatively low-cost and immediate, and the improvement in fuel efficiency can be up to 45%, which is also a significant reduction in carbon emissions. There are generally two ways of monitoring Eco-Driving; Using GPS and sensor data such as accelerometers and positioning from devices like smart phones, or acquiring data such as vehicle speed directly from the vehicle network. However, combining both would provide the most efficient results. As will be discussed, to extract information from the vehicle we use a combination of devices that capture raw vehicle data, convert it to readable format, and enriches with GPS information.

There is a large body of work that focuses on the benefits of eco-driving. The authors of [57] discussed the benefits of eco-driving in terms of cost savings and reducing emissions. It pointed out the primary advantages are that it can apply to any vehicle, across an entire fleet of vehicles immediately (as opposed to being phased in), resulting in immediate net savings to individuals from day one from greater fuel efficiency. The methodology of [55] consisted of installing in-vehicle monitoring technology via an OBDII port. The results show that gasoline and hybrid vehicles decreased average idling between 4% and 10% per vehicle per day, leading to an average emissions decrease of 1.7 kg of CO_2 per vehicle per day. The authors of [58] performed studies on the eco-driving skills of drivers. As a result of the training, reduction of fuel economy by 13.6% in average was achieved, which was reduced to 4% after a three month period. Similarly, [59] assessed driving behaviour of two groups. 853 drivers received training in eco-driving technique while 203 were monitored as a control group. This study found the driver education led to a statistically significant reduction in fuel use of 4.6% or 0.51 litres per 100 km compared to the control group. The above papers show the importance of monitoring driving behaviour in terms of fuel and CO_2 reduction. However, the methods of data extraction range from OBDII to using fuel-cards and paper forms for measuring fuel usage. Our goal is to evaluate OBDII and FMS standards over the same trips simultaneously, comparing them to actual fuel used, which we calculate from odometer readings and fuel used before and after a trip. Our aim is to give a clear indication of the most accurate standard for fleet telematics/urban mobility monitoring.

2.3.2 Controller Area Networks (CAN)

Modern vehicles are equipped with a highly complex network of electronic devices. Around 70 Electronic Control Units (ECUs) communicate with each other over the standard communication protocol known as Controller Area Network (CAN), Figure 5. Controller Area Networks were introduced by Bosch in 1991 to simplify the communication between different ECUs within a vehicles using one single pair of twisted wires, known as CAN-High and CAN-Low. The CAN-Bus is serial based, meaning the information travels along the CAN-Bus one bit at a time. On the CAN-High a redundant signal is transferred which is inverted compared to CAN-Low line. When CAN-High goes high, in the same time, CAN-Low goes low, in the same proportion [60].

CAN is a highly efficient protocol with the capability of providing a communication rate of up to 1Mbps [61]. CAN-Bus messages utilize a broadcasting system, allowing all ECUs to send and receive messages from the CAN-Bus. There are two well-known interfaces and standards that connect to the CAN-Bus; OBDII and FMS. While the CAN-Bus can be regarded as the networking system, OBDII and FMS can be viewed as languages that interpret the CAN data.

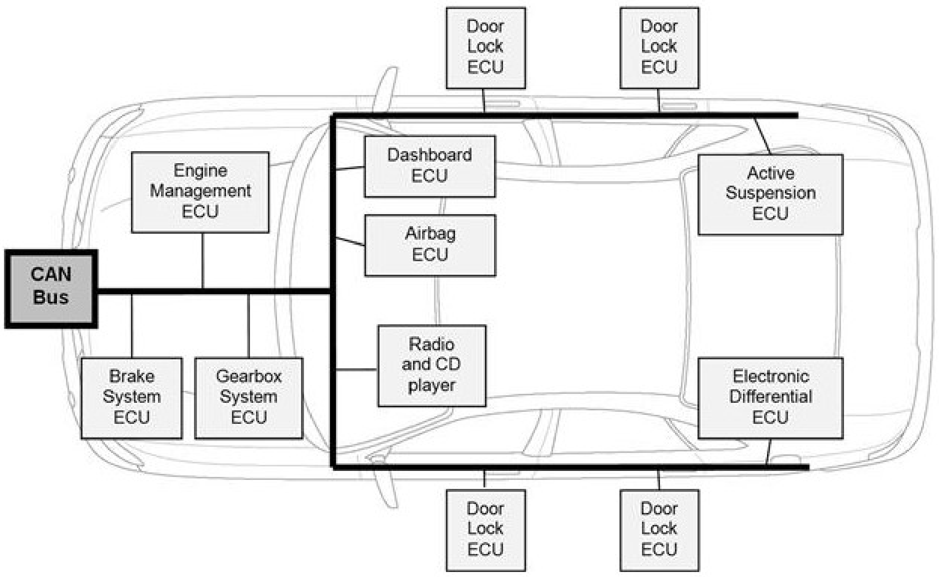


Figure 5: Example of CAN and ECUs within a vehicle

2.3.3 OBDII Standard

Since 1996 in the U.S and 2001 in Europe, all passenger vehicles come equipped with an OBDII (On-Board Diagnostics) port, usually located under the steering wheel. The standard was developed by the Society of Automotive Engineers (SAE). This specification was defined for all manufactured vehicles to enable the regulation of vehicle emissions, so as to ensure that the Environmental Protection Agency (EPA) standards are met [62]. The aim of OBDII was to provide communication to the CAN-Bus, giving access to real-time data from ECUs of the vehicle. The OBDII-II standard specifies the connector and its pinout, the electrical signalling protocols available, and the messaging format. Five of the 16 pins are standardized as shown in Figure 6. Pin 4 and 5 link to chassis ground and signal ground respectively, Pin 6 and 14 connect to CAN-High and CAN-Low respectively, and pin 16 connects to the vehicles power source. The remaining pins are vendor specific.

There are over 200 available parameters available via OBDII, which are called PIDs, and are requested via their hexadecimal identification number [63]. For example, vehicle speed has a PID of 010D. OBDII PIDs can be requested using an OBDII adapter, and sent to a smartphone using Bluetooth or Wi-Fi. Due to its ease of access, a large number of telematics companies identified the OBDII port as a way of obtaining vehicle data to support the delivery of new fleet telematics services and started to exploit it in their solutions [64]. However, as the name suggests, OBDII was not created for this purpose. Its original purpose was for mechanics to request diagnostic information. OBDII works on a request basis, with many experts regarding it as an intrusive way of gaining data.

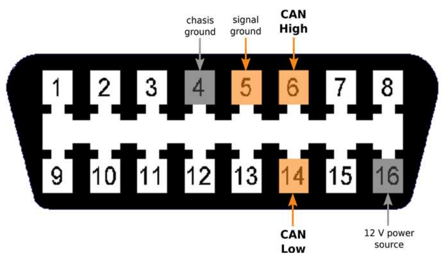


Figure 6: OBDII pinout

2.3.4 FMS Standard

In 2002, six major truck manufacturers (Volvo, Scania, Iveco, MAN, DAF, and Mercedes-Benz) decided to create a standardized vehicle interface for GPS based tracking systems, called the FMS (Fleet Management System) standard [65]. FMS is a subset of CAN-Bus data specifically created for fleet surveillance. There are currently over 30 FMS parameters that are specifically for fleet management. FMS is a high-sampled, reliable, wide-ranging and measured data provider solution. The data refresh rate is 10 milliseconds for direct CAN-Bus reading compared to the recommended 1-2 seconds for the OBDII port. FMS, until recently, was mostly available in trucks and buses, but the introduction of FMS gateways allows for the standard to be used in passenger cars and vans. Unlike its OBDII counterpart, FMS gateways have access to all CAN-Bus parameters, which, in some vehicles, totals over 2500. FMS has significant advantages over OBDII, in terms of security, data granularity, data precision, and it is far less intrusive as will be discussed.

2.3.5 Benefits and Limitations of OBDII/FMS

The benefits of OBDII is its ease of access and reduced installation time and cost. The OBDII port is easily accessible and there are a wealth of OBDII adapters and third party apps to avail of, Figure 7. However, [66] stated that using request-response with OBDII will force the ECU to enter a diagnostic session, thereby suspending other ECU's processes. The ECU provides priority to the latency sensitive, safety operations and stops OBDII's request-response queries. As a result, the overall query response time will be increased. The OBDII device is classed as intrusive in the sense that it repeatedly sends messages to the vehicle computer to request data. This becomes more problematic when the OBDII Bluetooth device is connected to a smartphone, and the internet. It is a potential safety and security weakness as passers-by, in theory, can connect to the OBDII adapter via Bluetooth and pass commands to the vehicle. Numerous vehicle companies such as BMW and Mercedes are opposed to the ease of access to the OBDII port.

"OBDII has been designed to service cars in repair shops. In no way, it has been intended to allow third parties to build a form of data-driven consumption on the access through this interface [64].

Some telematics companies acquire vehicle data directly from the CAN, connecting to the CAN-High and CAN-Low wires. However, a key challenge is doing so in a non-intrusive way [64]. Cutting or soldering connections to the vehicle CAN network can disturb its operation and cause damage. Recently, the development of a 'CAN-Bus clip', Figure 7, a non-intrusive reading device that clips to the outside of the cables, has addressed this issue. The clip does not puncture the protective sheath around the cables, instead, it 'senses' data passing through the wires, directly from the CAN network. In the image below, the yellow and blue twisted wires are CAN-High and CAN-Low (the four wires close together connect to the FMS gateway).

The CAN-Bus clip is safe, reliable and coded to understand each make & model. For this work, the clip is connected to a FMS gateway, which transforms the CAN signals into the FMS format. The cost of integrating an FMS gateway and clip in the vehicle is more costly than the OBDII option. It may also be necessary to require professional assistance in fitting the hardware in the vehicle. So, the decision in choosing which standard falls to the users' willingness to pay more for higher quality data.

Figure 7: Left; OBDII Adapter & Smartphone App. Right; Inventure Contactless Clip.

In general, a wide variety of vehicle sensors can provide valuable information. For example: a speedometer would be needed to detect a sudden decrease of speed, which could mean a danger of collision for the vehicles driving behind; when several vehicles detect that their average speeds are very low for a long time, it probably means that they are in a traffic jam; a substantial difference among the spinning of wheels could be due to the existence of sliding pavement; a deployed airbag could mean that the vehicle has crashed [67].

To date, the majority of literature has retrieved CAN signals via the OBDII port. Certain studies have compared OBDII data to GPS data when monitoring driving behaviour, regarding OBDII as ground truth. For instance, [68] performed a comparison between GPS and OBDII, reporting that GPS and smartphone sensor based techniques, combined with map and/or crowd-sourced data, can achieve greater than 94% correlation to OBDII information with regards to vehicle speed, acceleration etc. However, most modern telematics applications, and previous work such as [69], combine smartphone technology and internet access to collect and monitor driver behaviour retrieved via OBDII.

There are limitations in the literature regarding comparisons of vehicle data. A small comparison of OBDII and FMS was presented in [70], where the authors measure a limited number of parameters with the aim of choosing the best standard for a specific requirement. Vehicle speed and RPM were measured and compared, with only slight differences recorded. The real difference in FMS and OBDII was recorded when monitoring fuel economy. Testing shows the FMS fuel rate to be highly accurate compared to OBDII. However, only a basic algorithm using OBDII parameters was tested. The same authors again stated in [71] that when comparing OBDII and FMS, differences in vehicle speed and RPM can only be noticed in short measuring time (10-20 seconds). With regards to calculating fuel economy, corresponding calibration factors are required for OBDII. However, as previously mentioned, only a small number of parameters, and only one OBDII fuel algorithm was tested for a specific requirement. In comparison, our work evaluates numerous algorithms and a larger body of parameters from both OBDII and FMS. With the aim of comparing standards across numerous requirements in relation to Smart Cities.

A system was proposed in [66] to log OBDII data and direct CAN signals at the same time. This work primarily focused on the rates of data acquisition using a limited number of CAN and OBDII parameters for comparison. The OBDII adapter had a maximum request rate of 9Hz for one parameter, compared to 25 Hz for CAN. Although the authors state that the quality of analysis can be improved by having more information, results on improved accuracy due to an increase in data granularity were not provided.

Through combining the vast sensory information gathered from moving vehicles, with other Smart City experiments such as [36], [72], and [73], there becomes a wealth of information that gives invaluable insight into traffic behaviour, CO_2 emissions, pollution, temperature, humidity and so on. For these reasons, it is clear that CAN-Bus sensors hold significant importance and have a major role to play in the future development of Smart Cities.

2.4 WAVE Communication Protocol

Vehicle Ad-Hoc Networks (VANETS) [74],[75], [76] use a radio technology that allows vehicles in close proximity to exchange valuable information with one another, with the aim of enhancing road safety, traffic, and improve travel times. More recently, Internet of vehicles, [77], [78], [79] aims to integrate VANET technology with its surrounding environment, through the inclusion of RSUs. This is commonly referred to as V2I, or V2X [80], [48].

V2V and V2I rely on a technology called Dedicated Short Range Communication (DSRC) [81]. DSRC uses IEEE 802.11p, often referred to as Wireless Access for Vehicular Environments (WAVE). IEEE 802.11p was chosen for DSRC based on the fact that traditional IEEE 802.11 (a, b, g, n) are the most widely used wireless local area network standards in the world. Because of that, the cost of supporting equipment is low. 802.11p is better suited to the high mobility of V2V and V2I because of its changes in its MAC and PHY layers; i.e. discarding the need for authentication.

It is useful to appraise the differentiation between Basic Service Set (BSS), Service Set Identifier (SSID), and Basic Service Set Identifier (BSSID) to estimate the adjustments to the MAC layer introduced in IEEE 802.11p. A BSS is the fundamental part of the IEEE 802.11 standard and consists of a set of stations that can communicate with each other. An SSID represents the name with which the network distinguishes itself and is communicated, for instance, through a beacon frame by an access point (AP) in the case of BSS infrastructure. On the contrary, the BSSID recognizes the MAC address, composed of 6 bytes, of the access point. A new feature introduced in IEEE 802.11p is represented by the WAVE mode [82]. Whereas the traditional IEEE 802.11 standards connect to a Basic Service Set (BSS) via an access point, WAVE operates outside the context of a BSS (OCB) via a Provider/User scenario, the provider usually being an RSU. There is no MAC sublayer setup required before nodes exchange data frames OCB. The BSSID field of a frame sent OCB is set to all 1 s, i.e., 0xFFFFFF in hex notation, which is called the wildcard value.

The communication zone covered by a WAVE node is limited to 1km in line of sight, and less in non-line of sight, < 100m. When vehicles are moving at speed, this gives little time to exchange data, hence the need to discard authentication setup.

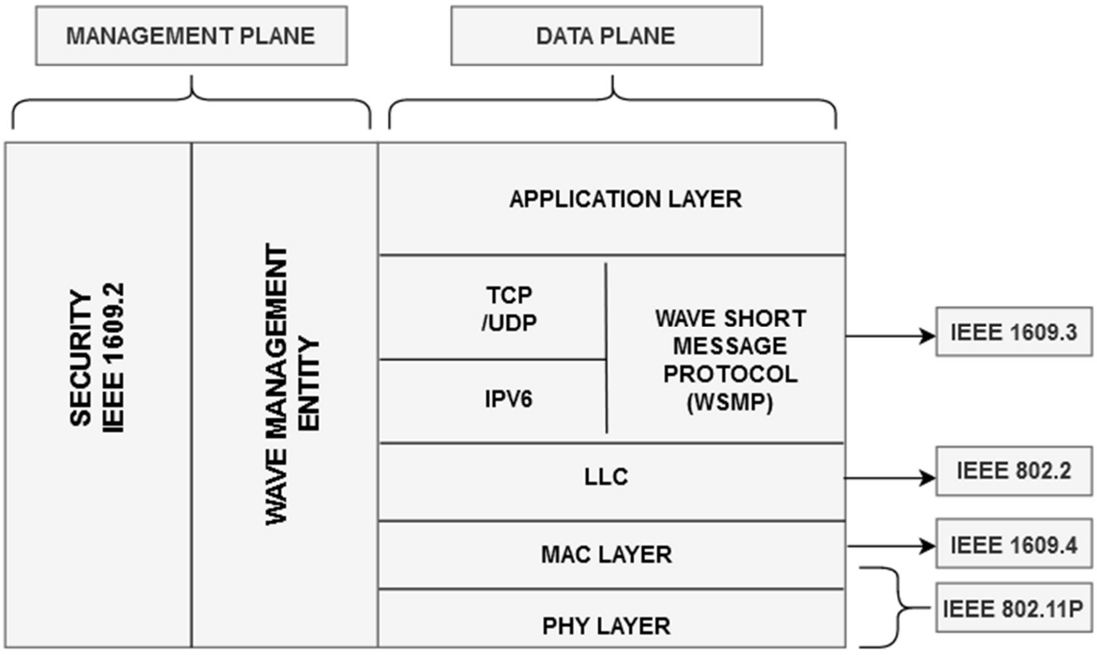


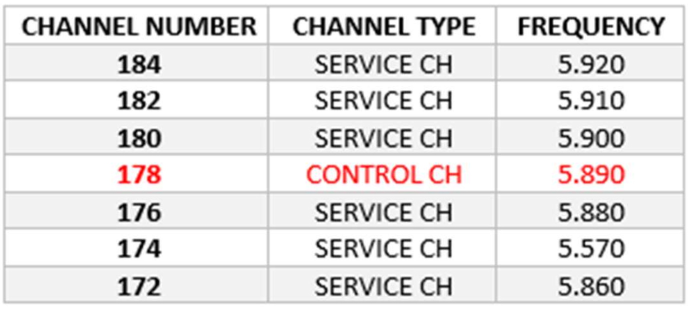
Figure 8: WAVE Protocol Stack and associated IEEE standards

As shown in Figure 8, WAVE supports IP based services along with its own WSMP (Wave Short Message Protocol). WSMP, as defined by IEEE 1609.3 [83] plays a key role in WAVE. The networking services provided by IEEE1609.3 are essential contributors to the low-latency and low-overhead characteristic. WSMP is specifically designed for the efficiency of WAVE devices in vehicular environments which allows applications to directly control physical parameters i.e., channel number, transmitter power and data rate used in transmitting messages [84]. There are two kinds of message transmission in

WSMP, Wave Service Advertisements (WSA) and Wave Short Messages (WSM) which will be discussed in the next section.

The main goal of WSMP is to support high-rate, low-latency communications between WAVE devices in a rapidly varying radio frequency environment. Management functions are performed by the WAVE Management Entity (WME) in which two WAVE device roles are defined. Devices transmitting WSAs which indicate the availability for data exchange assume the provider role, while those that can receive WSAs and have the potential to participate in data exchanges assume the user role. This information is stored in a Management Information Base (MIB) in the WME. Because there is no infrastructure, WAVE works on a channel switching basis. OBUs and RSUs transmit information through a 5.9 GHz (5.85-5.925 GHz) frequency band, which is divided into channels; a control channel and service channels; the CCH ID is 178 [81]. Table 1 shows the layout of the currently recognised channels.

Table 1: Current Control Channels and Service channel in the WAVE frequency range



CHANNEL NUMBER	CHANNEL TYPE	FREQUENCY
184	SERVICE CH	5.920
182	SERVICE CH	5.910
180	SERVICE CH	5.900
178	CONTROL CH	5.890
176	SERVICE CH	5.880
174	SERVICE CH	5.570
172	SERVICE CH	5.860

The default duration of the service intervals and control intervals is set to 50 ms, as recommended by the WAVE standard. Coordination between channels uses Coordinated Universal Time (UTC) for a global time reference provided by a global satellite navigation system. WSMP data is the only kind of data allowed to be transmitted on the Control Channel. All IPv6 data is restricted to the Service Channels.

2.4.1 Wave Service Announcements (WSA)

A WAVE node, which can be an On-Board Unit (OBU) or a Road-Side Unit (RSU) may advertise available services by way of sending periodic messages known as WSAs on the control channel. In turn, this node becomes a provider. Each WSA may include information needed to receive and process the information of the service being advertised, as well as the service channel to switch to. Nodes that are interested in this advertised information (Users), switch to the service channel provided, and begin exchanging information with the providing node.

In essence, a provider is creating a network on a specified service channel, and is advertising the network on the control channel via WSA. If the information being exchanged on the new network is of interest, a user will switch to that service channel and exchange information with the provider. Figure 9 shows a scenario between nodes exchanging data.

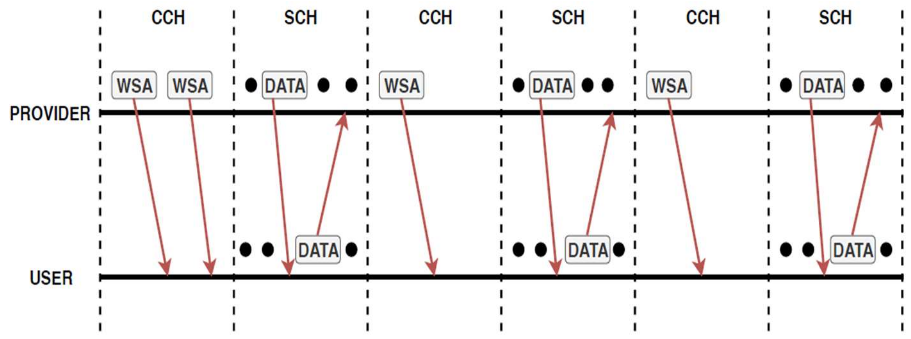


Figure 9: Provider/User exchanging information in WAVE during 50 ms intervals

Figure 10 shows the configuration GUI of a WSA using Estinet network simulator [85]. As shown in the image, the provider determines the service channel, data rate, transmit power, and priority of the application it is transmitting. Also included is a Provider Service Identifier (PSID), a globally unique value which resides in the header of the WSA. The PSIDs are designated to different services/applications, for example, safety messages are designated PSID1, Vehicle Telematics applications have been designated the PSID 3. If users are not configured to listen for PSID 3, they will ignore WSAs containing a PSID3 header. There is also another parameter called the Provider Service Context (PSC). This parameter is optional and can be used to provide additional information about the service. Currently, there is no literature that evaluates the role of PSC. However, this parameter will be implemented and evaluated as part of our city wide fleet management use-case, and also play a role in RSU governance.

If IP services are required, the WSA contains a Wave Routing Advertisement (WRA). The WRA contains the information needed by an OBU to access the Internet. The RSU will act as a gateway in this scenario, providing the OBU with its IP configurations.

2.4.1.1 Quality of Service (QoS)

Like its predecessors, 802.11p relies on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) for quality of service. Because all nodes have to share the same medium, two nodes cannot transmit at the same time. CSMA/CA provides the following structure:
A node that has a frame to send first senses the wireless medium. If the medium is idle, the node begins transmission of its frame. However, if the medium is busy, the node performs random back-off by choosing a number of idle time slots to wait before transmission.

Two QoS mechanisms in particular may be configured in WAVE scenarios; Enhanced Distributed Channel Access (EDCA) and RTS/CTS (request-to-send/clear-to-send). EDCA configurations may be included in the WSA. EDCA [86] is a QoS mechanism that can prioritize messages into categories based on importance. EDCA consists of four default Access Categories (AC), consisting of independent back-off entities to provide service differentiation. Each node supports four ACs with different QoS expectations. Consequently, outgoing packets are mapped to corresponding AC depending on their QoS requirements [87]. Prioritization in this access mode is reached by assigning different values of following contention parameters to each AC:

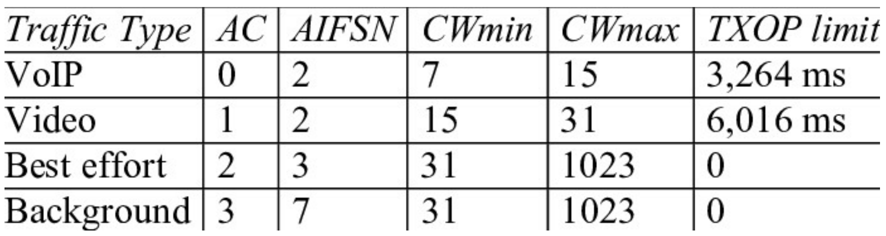
AIFS – Arbitration Interframe Space that defines the free time interval before back off stage

CW – Contention Window with its minimum and maximum size that provides the range of possible back off defer slots before starting the transmission. Similarly, with smaller maximum and minimum range higher priority is achieved.

TXOP – Transmission opportunity duration that specify a maximum time a station can designate on packet exchange. Thus, the larger TXOP duration the greater channel occupation possible.

There are four levels of priority in EDCA; Voice, Video, Best Effort and Background, Table 2. In the case of two data packets looking to transmit at the same time, voice would be given preference to transmit, whereas the other packet would back-off and wait for the next available slot for transmission.

Table 2: Default EDCA Parameters

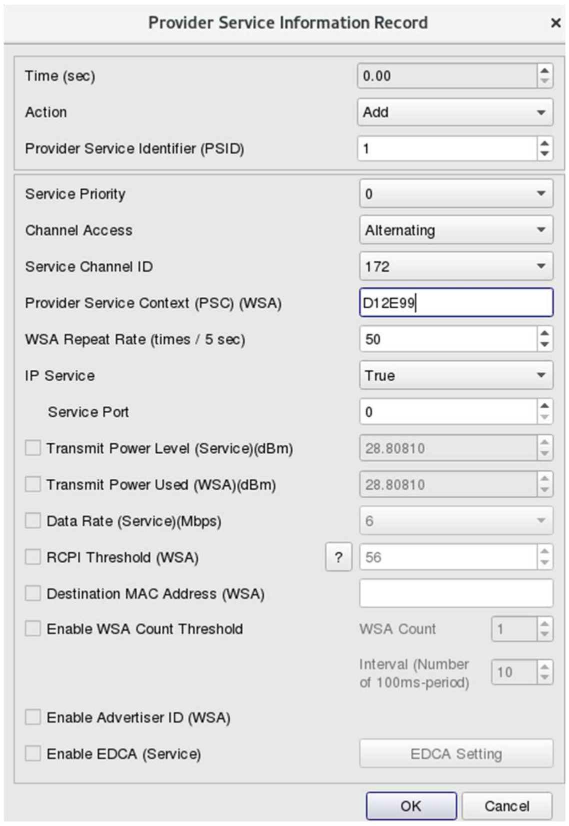


Traffic Type	*AC*	*AIFSN*	*CWmin*	*CWmax*	*TXOP limit*
VoIP	0	2	7	15	3,264 ms
Video	1	2	15	31	6,016 ms
Best effort	2	3	31	1023	0
Background	3	7	31	1023	0

As expected, there is a large body of literature on EDCA in V2V broadcast scenarios as the focus is with safety messages, which require priority over other packets. In a sense, a priority scheme like EDCA only makes sense if there is other traffic to give priority over, as such researching high priority settings in isolation makes little sense. However, this work aims to provide valuable insight in the operation of various contention window (CW) values in a unicast setting. Previous works such as [88] tested different CW sizes in a broadcasting scenario, coming to the conclusion that the contention window value of 15 is most adequate for safety beacons (15 falls under the video priority in EDCA). This will be taken into consideration in our simulations.

Another, albeit less reviewed QoE mechanism available is RTS/CTS [89]. The reason why RTS/CTS is less studied in V2V and V2I is because it doesn't apply to broadcast scenarios. However, as our focus is on unicasting, later chapters will evaluate the role it may play in improving distribution of our messages. The main goal of RTS/CTS is to address the "hidden node" problem, which is one of the main contributors to packet loss in vehicle scenarios. Because of the high mobility in vehicle scenarios, hidden nodes are a major issue. The problem occurs when two vehicles that may be in range of an RSU for example, but not in range of each other (they cannot sense each other) try to transmit at the same time causing packet collision.

Although 802.11p defines different data rates between 3 and 27 Mbps, 6Mbps was the data rate initially assumed in the standardization process, and since then it has been generally accepted as the default data rate. Higher data rates make use of high order modulation schemes and coding rates, and therefore require higher transmission power levels to reach a target destination node or area. This can be problematic in broadcasting scenarios, as higher transmission power can lead to more interference from further away vehicles. However, high data rates also reduce the packets' duration and therefore the channel load and interference. On the other hand, low data rates reduce the required transmission power levels to reach a target node. In contrast, they increase the packets' duration, channel load and interference. The authors of [90] presented the case for 6Mbps not being the optimal data rate for beaconing, reiterating the fact that high data rates reduce packets duration on the channel. For this reason, we will perform simulations using all available data rates in WAVE.


Provider Service Information Record

Time (sec)	0.00
Action	Add
Provider Service Identifier (PSID)	1
Service Priority	0
Channel Access	Alternating
Service Channel ID	172
Provider Service Context (PSC) (WSA)	D12E99
WSA Repeat Rate (times / 5 sec)	50
IP Service	True
Service Port	0
☐ Transmit Power Level (Service)(dBm)	28.80810
☐ Transmit Power Used (WSA)(dBm)	28.80810
☐ Data Rate (Service)(Mbps)	6
☐ RCPI Threshold (WSA)	56
☐ Destination MAC Address (WSA)	
☐ Enable WSA Count Threshold	WSA Count 1; Interval (Number of 100ms-period) 10
☐ Enable Advertiser ID (WSA)	
☐ Enable EDCA (Service)	EDCA Setting

Figure 10: Configuration example of a WSA using Estinet Network Simulator

2.4.2 Wave Short Messages (WSM)

The messages utilized by WSMP are known as WAVE Short Messages (WSM). WSM are designed to consume minimal channel capacity. The minimum WSM overhead is 5 bytes, and even with options and extensions it will rarely exceed 20 bytes. In comparison, a UDP overhead through IPv6 requires a minimum of 48 bytes. For many applications, especially in the active safety area, it is sufficient to exchange information in form of safety beacons between vehicles within immediate vicinity. In the EU, the beacon messages are referred to as Cooperative Awareness Messages (CAM), where as in the U.S they have been defined as Basic Safety Messages (BSM). There are slight differences between both message sets, however, they expect to be unified in the near future. From here, we will refer to these messages as beacons. Beacons are constructed from parameters directly from the vehicle CAN-Bus and enriched with GPS information. Table 3 shows the beacon message set. There are other parameters readily available to join the beacon;

for instance, if a driver hits the brakes, the Anti-Lock Braking System (ABS) parameter is added to the next beacon, alerting nearby vehicles of a sudden stop. Information stored within beacons provides a snapshot of current driver and traffic behaviour. Therefore, storing and aggregating this data can provide much insight to fleet managers about their fleets over a given time.

Table 3: Typical Message set of Beacons

Timestamp	3D Position	Position Accuracy	Speed	Heading
Heading	Vehicle Size	Steering Wheel Angle	Acceleration	Brake Status

2.4.3 IP Services

The choice between using WSMP or IPv6 depends on the requirements of a given application [2]. Single-hop messages, like those upon which collision prevention applications are based, typically use the bandwidth-efficient WSMP, while multi-hop packets use IPv6 for its routing capability. However, there is a limited amount of work that focuses on the implementation of IPv6 in WAVE. In [91] it stated that IPv6 works under certain assumptions for the link model that do not necessarily hold in WAVE. For instance, IPv6 assumes symmetry in the connectivity among neighbouring interfaces. However, interference and different levels of transmission power may cause unidirectional links to appear in WAVE, which may severely affect IPv6's effectiveness in its operation. Furthermore, it states that the current standard allows for a WAVE user to consume infrastructure based IP services only if there is a direct connection between RSU (i.e., WAVE provider) and WAVE user. Such condition is an undesired limitation of the WAVE standards. Other mechanisms are necessary to give access to the IP services to vehicles that exceed the one-hop range of WAVE users that do not directly hear the RSU.

An evaluation of IPv6 was conducted in [92]. The authors state that due to the diverse media involved in V2V such as DSRC, Wi-Fi, 3G, WiMAX, and LTE, IPv6 is the most promising technology that enables a convergence of such different communications over diverse media. This work investigates the issues regarding IP-communications over DSRC band. The investigation is made through field test experiments using communication devices equipped with hardware interfaces for different media as well as an IPv6 stack. Results showed it is possible to maintain stable communications between a vehicle and an RSU for longer than one kilometre of distance in line of sight scenario using a combination of a high power and a low data rate. However, this work was tested on a quiet road with just one vehicle.

Accommodating for IPv6 is further reiterated through the introduction of 5G based platforms. One such platform is CogITS [93]. The goal of CogITS is to address two challenges in implementing Intelligent Transport Systems (ITS).

1) Development of a reliable ITS communication network infrastructure
2) How to monitor V2V information remotely, or how to deploy the concept of live vehicles, in other words, the in-car real-time monitoring.

To address these issues, the authors propose a Cognition-enabled network management that utilizes machine learning techniques, towards an improved approach for the existing decision making logic, thanks to the data pattern techniques implemented inside the management solution. CogITS is comprised with six major parts:

Information Processing Manager, Decision Making Manager, Policy/Ontology Manager, Actuation Manager, Context Manager, and Device Manager. The context manager focuses on the V2V technologies such as WAVE and the context they provide, such as QoS and

Broadcast storms. WAVE-Flow is complimentary to platforms like CogITS, and may provide additional governance within the CogITS context manager.

2.5 Data Dissemination & Packet Loss

There is a large area of research that focuses on routing protocols and broadcasting in VANETS [74]. While routing algorithms are out of scope of our work, we will briefly discuss some of the known issues with routing and data dissemination in V2V and V2I scenarios. This will give us an idea on the expected packet loss rate in V2V/V2I scenarios. It is also important to make aware when multi-hop routing in VANETs is mentioned in current literature, they are relying on the IP services in WAVE as currently WSMP does not support multi-hop. However it must be noted that in Europe, 802.11p is used as a basis for the ITS-G5 standard, which supports the GeoNetworking protocol. GeoNetworking is a network layer protocol that provides packet routing in an ad-hoc network. It makes use of geographical positions for packet transport, and supports multi-hop communication.

There is a wealth of work focusing on routing protocols in WAVE. For example, [94] performed a comparative study on routing protocols in VANET. This paper reported the overall performance evaluation of two existing routing protocols namely, Ad hoc On-Demand Distance Vector (AODV) and Dynamic Source Routing (DSR). In their testing, using 60 vehicles, packet loss was 76% for DSR and 73% for AODV. Similarly, [95] and [96] reported packet loss within the same range for the same routing protocols for 50 vehicles.

One of the underlying issues with broadcasting in VANET scenarios is network overload and packet loss. This is due to a number of reasons, such as "Broadcast Storms", where vehicles retransmit incoming messages from other vehicles; in turn, causing the same messages being transmitted multiple times, congesting the network and causing high packet loss ratio (PLR). Another issue is the disconnected network problem, where there are not enough vehicles in close proximity to forward messages. There are many concepts to address the issues of broadcast storms and the disconnected network problems, however, there is a lack of technologies that address both [97].

There are two main methods for dissemination of safety beacons in VANET; the broadcast approach and Geocasting. The problem with the broadcast approach is not just broadcast storms, there is also the issue that messages are sent to all vehicles without exception, which is useless in many scenarios. There are many techniques adopted in the literature to improve this issue. For example, one such technique is called slotted persistence [98], which is a TDMA inspired approach in which different waiting time slots are assigned to vehicles depending on their locations. A shorter waiting time is assigned to the vehicles located furthest from the broadcaster.

Another dissemination technique is Geocasting [99]. Geocast protocols aim at disseminating information only to vehicles inside a specific geographical area, called ZOR or Zone of Relevance. This is can be seen as a more feasible approach as safety events are only of interest to vehicles in a specific location. Without leaving the scope of this work, we will address [100] which proposed a novel distributed beacon scheduling scheme referred to as the context awareness beacon scheduling (CABS) which is based on spatial context information dynamically scheduling the beacon by means of TDMA-like transmission. The proposed beacon scheduling scheme was evaluated using different traffic scenarios within both a realistic channel model and IEEE 802.11p PHY/MAC model in their simulation. The simulation results showed that the performance of the CABS scheme was better than periodic scheduling in terms of packet delivery ratio and channel access delay. CABS along with two ray ground propagation model improved packet loss for 300 vehicles from 80% to 50%.

The issues mentioned above are not resigned to V2V. V2I scenarios also suffer from network overhead and high packet loss. Certain applications, particularly non-safety applications may be unicast; i.e., transmitting to a specific node, an RSU for example. This also causes issues as the RSU is receiving a large quantity of information simultaneously from a large amount of vehicles, resulting in collisions and packet loss. Also, the hidden node issue can affect QoS in unicast scenarios in V2I, as multiple vehicles out of range of each other may attempt to transmit to the RSU at the same time.

2.6 Flow Based Programming

Implementing the appropriate programming model should not be overlooked when creating a solution for data processing. Developers have to address issues in relation to dynamically configuring and managing data processing tasks over Cloud and Edge devices, and optimizing task allocation for minimal latency and bandwidth consumption. Although the majority of programming in the data processing world has its roots in the procedural programming model, our architecture is based on a model that is beginning to make its way into the IoT mainstream.

Flow based programming (FBP) [101] is a model ideally suited to IoT. The aim of FBP is to concentrate on the data and data streams first before deciding what processes are needed to convert between the different data streams. In Object-orientated programming, you have to decide on the object classes, and then decide what messages each class should be able to respond to [102]. While it is possible to create real time interactive IoT applications using traditional programming tools, it can quite often be a difficult task. Developers are required to learn new protocols, while creating data processing components, and link them together. In recent years, several runtime environments such as [103], [104] and [105] have begun to implement FBP inspired approaches. One major advantage to FBP, although not necessarily expressed visually, lies at the heart of several visual programming languages. Visual data flow programming languages (VDFPLs) [106] have been used in many other domains such as high performance parallel computing, leveraging multi core processors, music and industrial applications [107] . FBP is best understood as a coordination language opposed to a programming language.

One of the prime advantages of FBP is its modularity, meaning the degree to which a system's components may be separated and recombined. Nate Edwards of IBM [108] coined the term "configurable modularity" to denote an ability to reuse independent components just by changing their interconnections. A main characteristic of systems that exhibits “configurable modularity” is that you can build them out of "black box" reusable modules. While it is necessary to connect them together, they do not have to be modified internally to make this happen. FBP has three main components:

Black Boxes: Each black box, or process, in the application is an instance of a component that essentially receives some data, processes it and forwards the output to another black box, creating a dataflow.

Bounded Buffers: These are the connections between the black boxes. Black boxes are connected to one to another through ports defined by their components. The black box receives data through an input port and transmits the result through an output port. Black boxes may have multiple input and output ports. It does not need to know anything about the other black boxes are connected to those ports.

Information Packets (IP): IPs are the data that travels through the network. IPs are most often in the form of structured packets or streams of packets. They can be owned by only one black box at a time, which will either pass it along to the next process in the network or drop it. Figure 11 shows a basic dataflow between an Edge device and the Cloud, consisting of numerous black boxes, connected via bounded buffers.

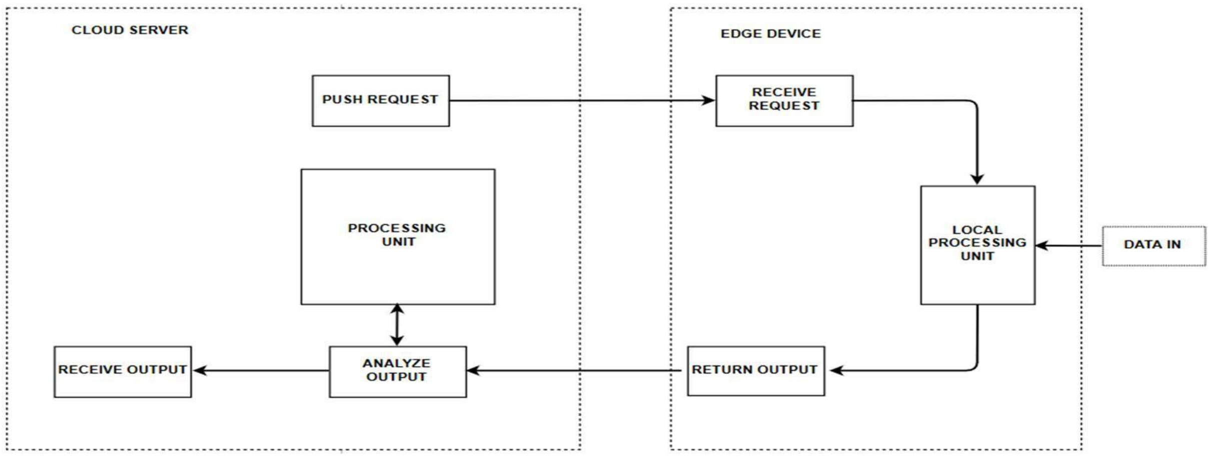


Figure 11: Basic FBP dataflow between Edge Device and Cloud Server

The modular fashion of FBD would ease the distribution of the flows from the Clouds to the IoT devices leveraging the Edge-Computing concept. As previously mentioned, there are a growing number of recent works beginning to implement FBP. Node-RED [105] is an open source project created by IBM. It allows the user to build IoT applications and services, but has been designed as a run-time for individual services. In [109], the authors examined the development of IoT applications from the perspective of the Fog-Computing paradigm. The authors implement a Distributed DataFlow (DDF) programming model that utilises computing infrastructures across the Fog and the Cloud. The framework is based on Node-RED (NR). The authors extend NR with their creation Distributed Node-RED (DNR), to ease the development process while allowing developers to leverage resources between devices and the Cloud to build Fog based IoT applications. While this model provides an easy way to design and develop IoT applications, there are several open issues. To facilitate the communication between devices spanning multiple networks, it may be necessary to include a distributed discovery and communications infrastructure. In addition, currently there is only a prototype design and early work is discussed without any platform readily available.

UFlow was proposed in [110]. UFlow is a concept of data flow transformation closer to the source, on the devices with constrained resources. The authors analysed two tier

IoT architecture composed of devices and the Cloud. The scientific contribution of the paper as well as the concept of the data flow transformation is that the UFlow framework can be executed on a variety of resource-constrained embedded devices, and can be implemented on a NodeRED platform. FogFlow is proposed in [31]. The aim is to ease the service orchestration and scalability for geo-distributed Smart Cities. Through the implementation of the dataflow programming model, developers in FogFlow only define a service topology and decompose the IoT service into multiple processing units (black boxes).

In relation to Fog-Computing, [111] cited that how to efficiently deal with dynamic variations and transient operational behaviour of Fog-Computing is a crucial challenge within the context of choreographing complex services. Furthermore, with the rapid increase of the scale of IoT deployments, the heterogeneity, dynamicity, and uncertainty within Fog environments and increased computational complexity further dramatically aggravate this challenge. Taking this into account, it is this author's opinion that the format of FBP and how it can process data in motion, including converting data types quickly can address complex issues such as stated above.

2.7 Discussion

The aforementioned papers have provided valuable insight about the current state of the art. As discussed, there is a large body of relevant work in this area to build on. This section showed how and why there is a recent paradigm shift from the Cloud centric approach to Edge and Fog-Computing. What is apparent throughout the literature is the lack of platforms for developers to deploy and maintain IoT applications on Edge/Fog nodes. This statement is backed up in recent survey papers [32], [5], [23] and [112].

This chapter has shown the main difference in functionality between Edge and Fog lies in two areas; locality of processing, and networking between multiple devices. In many scenarios, Fog will be implemented on infrastructure nodes whereas Edge-Computing will be implemented on infrastructure-less nodes. In the near future, the difference between the roles that Edge and Fog-Computing provide may become more transparent with the adoption of larger IoT applications such a Smart Cities and Connected Vehicles. It is apparent that any architectural proposals should have the capability to allow coordination between Edge/Fog and Cloud-Computing. Due to the lack of compute power on some Edge devices, it may be necessary to split the IoT applications into multiple tasks, with the heavier computational tasks performed at the Fog or Cloud layers. This, in effect, should reduce latency for the more mission critical tasks. This area will also be taken into account in the following chapters of this thesis.

This chapter has shown that vehicles have potential to be a key enabler in the development of Smart Cities due to the wealth of sensor data, and also their mobility throughout the city. It is also imperative that vehicles have strong processing and communication capabilities. The body of work presented in this chapter has strengthened our focus on answering the research questions presented in chapter one. Although we have discussed multiple bodies of work that present Edge processing capabilities, there seems to be a lack of bi-directional communication that allow developers to make instantaneous changes in the local processing on Edge nodes. Furthermore, in terms of vehicles in urban areas, it is our goal to enhance RSUs from access points to governing bodies in a V2I scenario, while focusing more on bidirectional communication between OBUs and RSUs that is presented in the current state of the art.

Because of the limited work on IPv6 in WAVE, along with the fact IP services may be a valid alternative for distributing vehicle sensor data in V2I, we will dedicate a section of our results to evaluating IP in our simulations. Most vehicles in the near future will consist of multi-interfaces (802.11p and 802.11 a/b/g/n), so it is important to perform an evaluation on IP, using results mentioned in the packet loss section above as a baseline.

To address the issue of RSUs receiving too much data at any given time, we introduce a mechanism that allows the RSU to modify the granularity of packet inter-arrival times from the vehicles. QoS mechanisms will also be evaluated within our architecture. EDCA is widely used in broadcast scenarios, giving priority to safety messages, however RTS/CTS may be more applicable to our use case. It is our aim that a combination of QoS mechanisms, increasing the data rate and granularity may improve the distribution of our messages in the use case. To finish this section, it is no coincidence that there is a recent growth of interest in flow-based programming. Reviewed papers in this section, along with the number growing applications in industry that use this programming model strengthen its case for use in IoT. For this reason, it will be the programming model of choice for the proposed architecture.

3 Distribution And Governance of LArge Datasets (DAGLADS)

3.1 Architectural Design & Methodology

There are several design goals for an adequate reference architecture. The definition of a "Distribution And Governance of LArge Data Sets" (DAGLADS) Reference Architecture (DRA) is built on existing standardisation efforts in order to specify a high-level architecture, DRA, for the system. In this way, the requirement for in-depth knowledge of the state-of-the-art is addressed. As pointed out in the previous chapter, there is a lack of platforms for IoT applications on Edge/Fog devices. With this in mind, the goal of this chapter is to define the fundamentals required for our reference architecture, and present the methodology to meet said requirements. This section will also define the entities/nodes and their relationship within the architecture. Within each node, system components and accompanying dataflows will be detailed, giving an overall view on the communication and collaboration between nodes. Finally, this chapter concludes with implementations and results of the reference architecture showing its capability in an IoT paradigm.

Building on knowledge gained from the literature and using current state-of-the-art as a reference, this chapter will provide a discussion on what methodology is required to answer the following research questions.

Is it possible to design a data-centric architecture with the necessary performance to support bi-directional communication between Edge/Fog/Cloud nodes in an IoT paradigm?

In terms of communication and collaboration between nodes, to what degree can Flow Based Programming address the heterogeneity of services and applications in a Vehicle-to-Infrastructure scenarios (V2I)?

3.1.1 Defining the Reference Architecture

In order to achieve the research questions, first we must consider the following requirements and fundamentals:

Latency Sensitivity – The driving force behind Edge Processing is to reduce latency. The proposed architecture must be capable of offering the end users low-latency-guaranteed applications and services. Local processing reduces the execution time of a task and increases the speed of decision-making.

Generality - Due to the heterogeneity of Edge/ Fog nodes, the architecture must be capable of handling a diversity of data types, and data generation rates.

Efficiency – A set of rules must be set to monitor and efficiently utilize resources on the Edge device. The reasons for this are that Edge/Fog devices may have limitations in compute power, memory and storage.

Programmability – Developers should have the ability to deploy and modify the applications remotely. This will require an efficient bi-directional communication system between Edge devices and the Cloud.

The following Edge capabilities are crucial to a business's IoT success:

Out-of-the-box bidirectional connectors

Remote management for Edge devices

Data processing capabilities at the Edge

Self-contained Edge nodes

3.2 Architectural Software Overview

3.2.1 Programming Model

To address the above requirements and necessary capabilities, our first step was to choose an efficient programming model. IoT software must be efficient in a dynamic environment and have the ability to seamlessly deal with many concurrent inputs and outputs. Handling large amounts of interactions between IoT things will prove to be a very challenging task. It will be necessary to have in place an appropriate organization of data processing. The traditional way of implementing processing logic is often based on Service Oriented Architecture (SOA), where devices expose their functionalities as services, which are then consumed by the ones deployed in the Cloud [8]. However, the velocity at which data is created in an IoT environment can make that approach insufficient [110].

Our literature review has shown that Flow Based Programming (FBP) can be viewed as a technology where an application is constructed as a network of asynchronous processes exchanging data chunks and applying transformations to them. The creator of FBP proposed that developers spend less time thinking about the order in which things are executed (control flow), and more time focusing on the data and the transformations that are applied to it (data flow). The goal of FBP is that application development has a more natural flow to it. Although first created at IBM in the late 1960s as a software development paradigm, there has been a noticeable increase in technologies inspired by the FBP paradigm recently. Authors of [113] discuss many of the inherent benefits with the data flow /flow-based programming paradigm, including implicit pipeline parallelism, exceptional composability, testability, inspectability and code re-use. Projects such as NoFlo, NodeRED, Apache Nifi, and Cisco Kinetic have begun to focus on the strengths of FBP and the processing of data flows, which is a major requirement of the modern data-driven applications, thus making it a viable programming model for this oncoming paradigm shift.

3.2.1 *Flow Based Programming Technology*

Apache Nifi [114] is a data in motion technology that uses flow based processing. The main goal of Nifi is to automate the flow of data between systems. Nifi provides a user friendly GUI and contains over 200 processors. The user can create a real time dataflow by dragging processors onto the canvas. Each processor is configured by the user to perform a specific action on the passing data. Each piece of data flowing through the Nifi dataflow is referred to as a flowfile. The built in Nifi processors can be configured to perform a wealth of actions such as converting data formats, adding attributes to the data, and routing data based on attributes. There is also a collection of processors available for ingesting data from a multitude of sources including URLs, ports, databases, local file systems, and external sources such as Edge devices. Nifi is one of the main components of the proposed architecture and has often been associated with the term "The Swiss army knife for IoT". Figure 12 provides an example of the Nifi canvas with a group of processors connected to form a dataflow. Table 4 describes the Nifi phrases and the FBP counterparts.

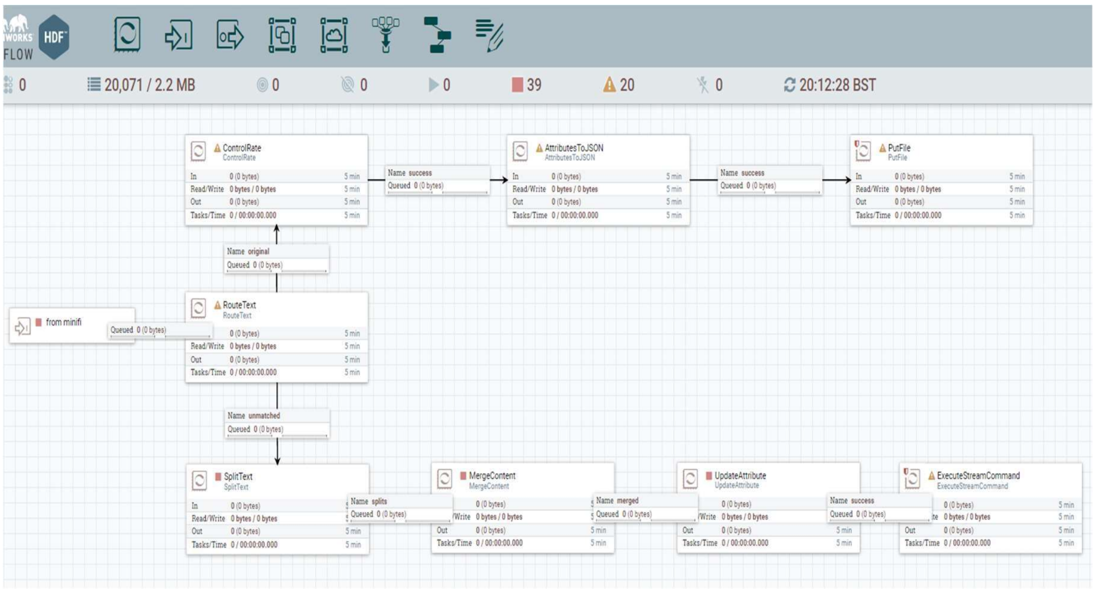


Figure 12: Example of Nifi Dataflow and a collection of connected processors

Nifi was created by the National Security Agency (NSA) and acquired by Hortonworks. Nifi addresses many of the technical challenges associated with IoT, such as adding extra security to the transportation of data with built-in support for SSL, SSH, HTTPS, encrypted content and role-based authentication/authorization and handles a diversity of datatypes as described above. Apache MiNifi [56] is a sub project that can perform almost all of the actions Nifi can. MiNifi does not come with a GUI and is much more lightweight and optimized to perform on smaller Edge devices. Dataflows are created on the central Nifi server and downloaded onto the Edge devices featuring MiNifi.

Table 4: Comparison of Nifi Terms and FBP Counterparts

Nifi Term	FBP Term	Description
Flowfile	Information Packet	A FlowFile represents each object moving through the Nifi system.
Processor	BlackBox	Processors perform the work on the passing flowfile. A processor is doing some combination of data routing, transformation, or mediation between systems. Processors have access to attributes of a given FlowFile and its content stream.
Connection	Bounded Buffer	Connections provide the actual linkage between processors. These act as queues and allow various processes to interact at differing rates. These queues can be prioritized dynamically and can have upper bounds on load, which enable back pressure

Nifi can also be seen as an advanced ETL tool, (Extract, Transform, and Load). The ETL process became a popular concept in the 1970s. Data extraction is where data is extracted from data sources; data transformation is where the data is transformed for storing in the proper format or structure for the purposes of querying and analysis; and data loading is where the data is loaded into the final target database. Figure 13 shows an ETL dataflow, where operations are performed on the data before it is finally stored. Apache Nifi has the capability to perform the transformations while the data is in motion. For our work, Apache version 1.8.0 was used.

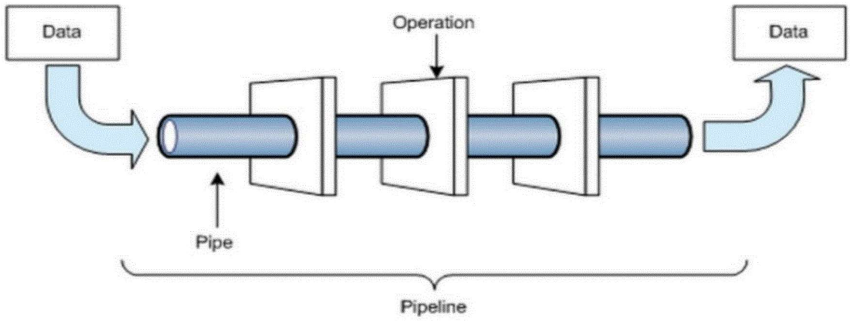


Figure 13: Typical ETL scenario

3.2.2 Cloud Platform

Current Hadoop distributions are provided by data analysis companies, with the top three being Cloudera, Hortonworks and MapR. These distributions are open source and provide businesses with a platform containing multiple technologies for data processing, analysis and storage. However, although these distributions are regarded as current state of the art for data analysis, they are completely centralized. Hortonworks, however, has released a platform called Hortonworks dataflow (HDF) which includes software that allows companies to begin processing data closer to source. HDF is a suite of tools that give the user full control of data from its generation on the Edge devices, while solving the real time challenges of collecting multiple types of data from a multitude of sources.

The three applications within HDF are Apache Nifi, Apache Kafka and Apache Storm. Apache Nifi is a fundamental part of the proposed architecture and will be discussed in detail in the following section. Apache Kafka [115] is used as the messaging service as it provides high throughput, reliable delivery, and horizontal scalability. Kafka is a low latency-messaging platform for real-time streaming data sources. Within Kafka, there are four main components; Producers, Consumers, Topics and Brokers. Kafka messages are organized into Topics, which are a category or feed name to which records are published. A producer pushes messages to a specific topic; a consumer pulls messages from a specific topic. Kafka runs in a cluster, as it is a distributed system, and each node in the cluster is known as a broker.

Apache Storm is a distributed computation system that performs real time processing on large amounts of data. There are five key elements of Storm; Tuples, Streams, Spouts, Bolts and Topologies. A Tuple is a list of ordered elements. Generally, it is a set of comma-separated values. A stream is an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion. Spouts are the source of data; in this case, they will be the Kafka Topics. Bolts are the process units. They process incoming streams and produce output streams. Topologies can be viewed as a network of spouts and bolts.

The following section will describe the main technologies used in the creation of the proposed architecture.

3.2.3 Data Science Platform

Anaconda is a python based Data Science platform [116], which comes with over 700 python and R packages. The Anaconda platform is used throughout this work to create all learning models and python codes that are executed. The primary library used for this work is Scikit-learn [117] a machine-learning tool built on NumPy, SciPy, and Matplotlib. All of the technologies used are open source and readily available. Anaconda 5.0.0.1 was used for this work, along with NumPy 1.1.3.1 and Scikit-learn 0.19.0.

3.2.4 Automated ML Technology

TPOT [118] is a Python Automated Machine Learning (AutoML) tool that optimizes machine-learning pipelines using genetic programming. TPOT was an additional package that was installed on Anaconda and can be viewed as a Data analyst assistant. It works by intelligently exploring thousands of possible pipelines to find the best one for your data. Once completed, it provides a python code to build the model with the most optimized hyper parameters. AutoML algorithms are not as simple as fitting one model on the dataset; they are considering multiple .machine learning algorithms with multiple pre-processing steps. With the default settings, TPOT will evaluate 10,000 pipeline configurations before finishing. The user can alter these parameters to perform a quicker search if necessary. Figure 14 represents the area automated by TPOT.

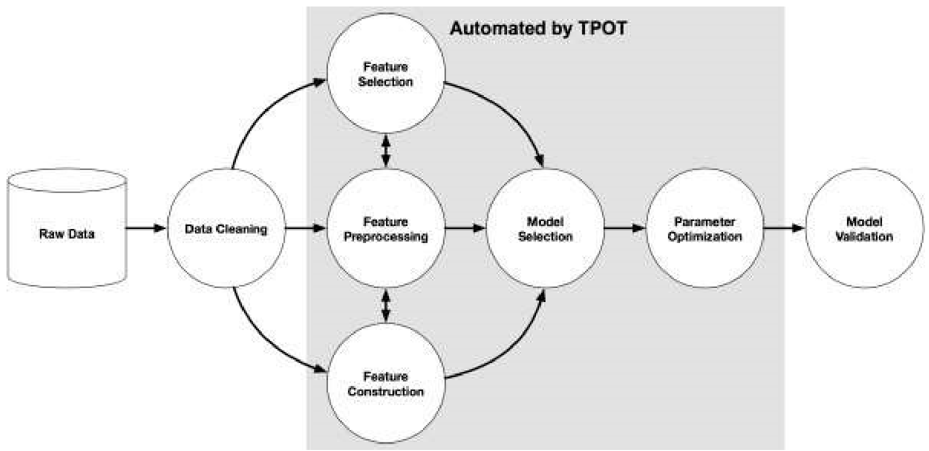


Figure 14: Area in which TPOT aims to automate

3.2.5 Service User Interface

Netcat is a computer networking utility tool designed to read and write data across TCP and UDP connections using the TCP/IP protocol. It is often referred to as a Swiss army knife utility due to its versatility. Netcat is used as a service UI throughout this work, allowing the developer to push requests into the dataflows in real time and view the output through specified ports.

3.3 Architectural Entities

The DAGLADS architecture consists of three main entities:

IoT Sensor Device – The IoT devices are the sensor devices that create the data. In many scenarios, these will have limited resources, and transmit data to an Edge gateway device for processing.

Edge Container – The Edge container represents an Edge device with local processing capability. In some cases, it may represent the device generating the data, in other cases it may represent a device acting as a gateway for less powerful IoT sensor devices. Data is ingested from the IoT devices and is processed locally or offloaded to the Cloud.

Central container. The Edge container may be viewed as an agent to the Central container. The Central container is a Cloud server with large processing and storage capacity. A service UI allows the developer to push control information into the dataflows between the Edge and Central containers in real time.

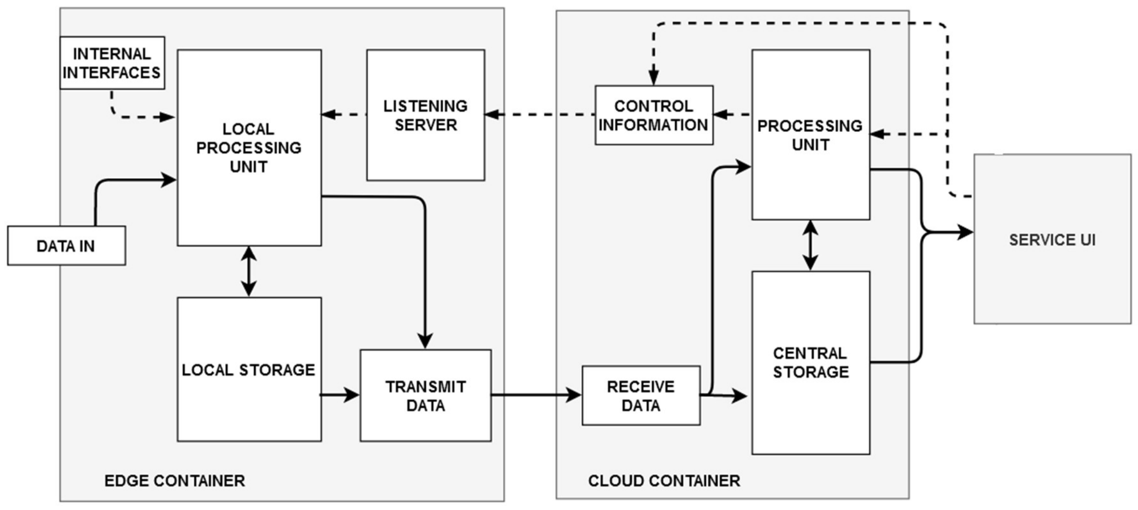


Figure 15: Reference architecture showing dataflows between Edge&Cloud. Solid lines represent sensor data moving through Edge&Cloud components. Dotted lines represent control information passed to the local processing unit.

Figure 15 represents the data movement between components of the reference architecture. Solid lines represent data movement whereas dotted lines represent control information. Through the service UI, the developer can seamlessly modify, in real time, the data analytics performed on the Edge and Central container. This is achieved by passing control information into the dataflow, which in turn, is transmitted to the Edge containers. Data received from the Edge containers may also be viewed in real-time through the service UI.

The Central container ingests data from the Edge, where it is routed to a central database for storage, or the processing unit for further analysis. The processing unit performs model building as it has access to a learning algorithm repository and the local database. Models may be implemented on the central container or pushed out to the Edge containers. The processing unit also performs data mining, in real time and/or batch. The Central container acts as a coordinator for the Edge devices/agents with the ability to send control information to its agents. This information is determined by external factors or user requests directly from the service UI.

The Edge container ingests data from the IoT devices and passes it to the local processing unit. Developers' generic applications are incorporated into the local processing unit. Data analysis may be influenced by internal factors such as network connectivity, CPU or RAM usage, external factors such as bandwidth, or requests received from the central controller. This authors publications [119], [120], and [121] were devoted to various implementations of the DAGLADS architecture.

3.4 Edge & Cloud Components

Building on the reference architecture shown in Figure 15, this section further breaks down the components on the Edge and Cloud nodes that allow for bi-directional communication and collaboration. For example, within the local processing unit on the Edge node, are a number of components that create a task deployment scheme for monitoring system and energy consumption. This allows for a collaborative protocol between the Edge and Cloud, where computation may be offloaded to the Cloud to preserve local resources.

3.4.1 Edge Container Components:

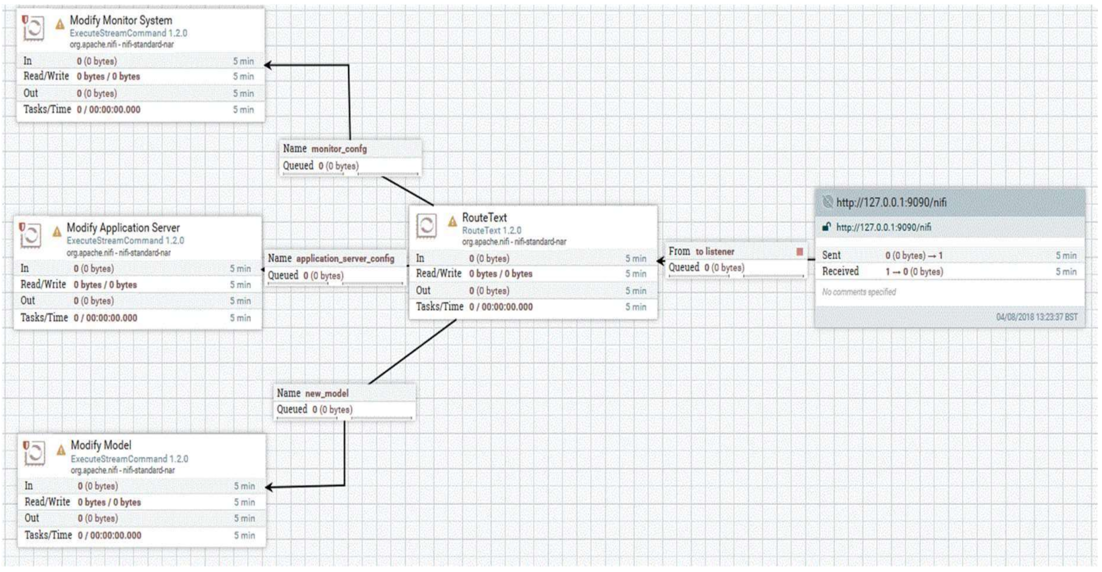
System Monitor: A monitoring system regularly checks, at set intervals, the main system components such as CPU usage, RAM usage, network connectivity, and storage. The developer, via the Central container can modify the thresholds. When the thresholds are exceeded, an alert is written to a monitoring.txt file and is detected by the Application Server.

Application Server: Performs processing on incoming sensor data and stores the output locally, or returns to the central container. The server regularly checks alerts from the monitoring.txt file at set intervals, to determine what processing occurs locally. Custom software is in place that will offload computations to the Central container based on the alerts. The application server is configured to ingest data via a specified TCP/UDP port. Flowfiles are passed into the application server via FBP processors such as PutTCP, or PutUDP. For this reason, the application server is used where a higher velocity stream of data occur, or mission critical data is processed.

Local Storage: In most cases, local storage may be limited on Edge containers. The system monitor regularly checks the storage capacity on the Edge container. When a disk-usage threshold is exceeded, a custom mechanism compresses and offloads data from the local storage to the Central container. Blocks of data are offloaded at set intervals until a minimum disk-usage threshold is reached. For example, when the system detects storage has passed 80% disk-usage, data is compressed and offloaded to the Cloud in 20MB blocks at set intervals. This may also be configured on a time-of-day based scheme, whereas data is offloaded in off-peak hours such as 12.00 a.m. – 4.00 a.m. daily.

Listening server: Ingests control information from the Central container. Incoming information is assessed with custom protocols that will modify system components based on the received information. The incoming information may consist of new applications, models, or parameters to be changed within the Edge container system components.

Figure 16 shows the dataflows within the listening server. A RemoteProcessGroup processor is configured to ingest information from the Central Controller. The RouteText processor is configured to filter and direct the incoming information to the waiting applications. Each dataflow contains an ExecuteStreamCommand processor that will run the necessary requirements to address the modifications. Custom applications are configured to modify specific components in the Edge container. Figure 17 provides a pseudocode example of the protocols in place.



Modify Monitor System
ExecuteStreamCommand 1.2.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name monitor_config
Queued 0 (0 bytes)
Modify Application Server
ExecuteStreamCommand 1.2.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
Name application_server_config
Queued 0 (0 bytes)
RouteText
RouteText 1.2.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min
From to listener
Queued 0 (0 bytes)
http://127.0.0.1:9090/nifi
http://127.0.0.1:9090/nifi
Sent 0 (0 bytes) → 1 5 min
Received 1 → 0 (0 bytes) 5 min
No comments specified
04/08/2018 13:23:37 BST
Name new_model
Queued 0 (0 bytes)
Modify Model
ExecuteStreamCommand 1.2.0
org.apache.nifi - nifi-standard-nar
In 0 (0 bytes) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 0 (0 bytes) 5 min
Tasks/Time 0 / 00:00:00.000 5 min

Figure 16: Listening Server Dataflow- Remote processor group (far right) passes information received from the central controller into the dataflow

```
Pseudocode for Listening Server

If incoming = *new_model.*
Replace existing model with .*new_ model.*
Restart Application Server
If incoming = *new_application.*
Application_Server = *new_application.*
Restart Application Server
If incoming = *CPU_Threshold*
Restart System Monitor
Where CPU_Threshold = new CPU_Threshold
```

Figure 17: An example of the protocols in place on the listening server

Apache MiNifi: Allows for bi-directional communication between the Edge and Central container. MiNifi is also configured to move data throughout the Edge container. Sensor data is ingested and passed into the application server, while control information is ingested from the Central container and passed into the listening server, as shown in Figure 18.

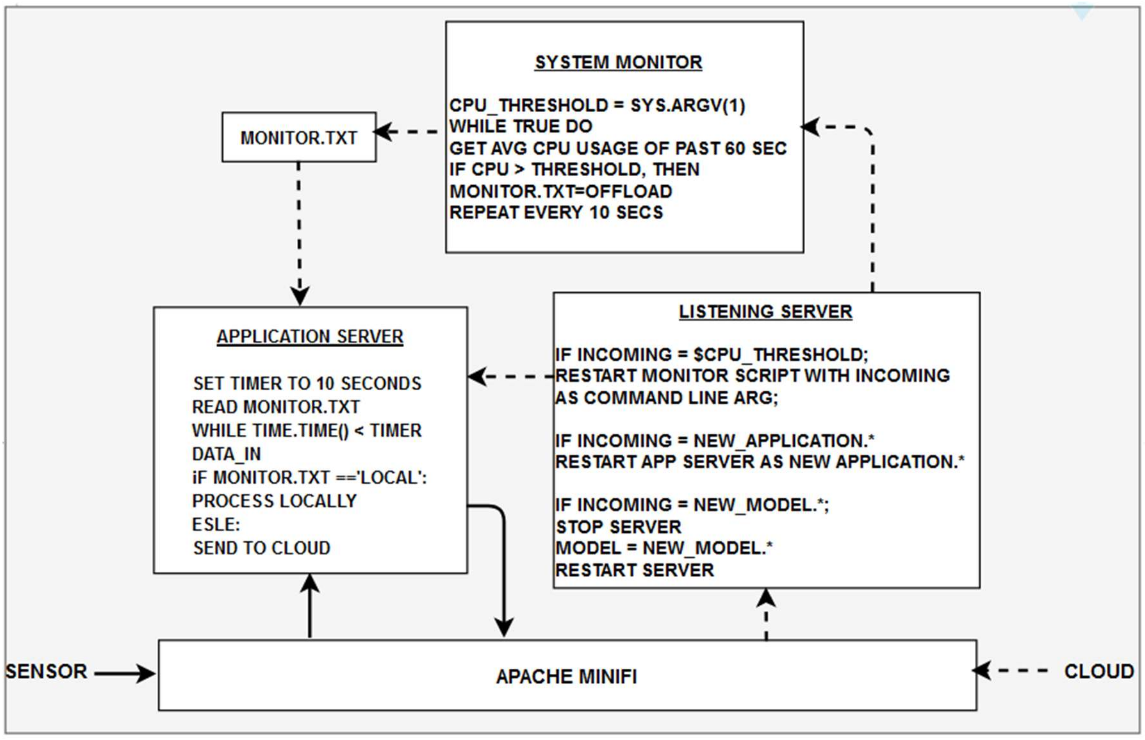

SYSTEM MONITOR
CPU_THRESHOLD = SYS.ARGV(1)
WHILE TRUE DO
GET AVG CPU USAGE OF PAST 60 SEC
IF CPU > THRESHOLD, THEN
MONITOR.TXT=OFFLOAD
REPEAT EVERY 10 SECS
MONITOR.TXT
APPLICATION SERVER
SET TIMER TO 10 SECONDS
READ MONITOR.TXT
WHILE TIME.TIME() < TIMER
DATA_IN
iF MONITOR.TXT =='LOCAL':
PROCESS LOCALLY
ESLE:
SEND TO CLOUD
LISTENING SERVER
IF INCOMING = $CPU_THRESHOLD;
RESTART MONITOR SCRIPT WITH INCOMING AS COMMAND LINE ARG;
IF INCOMING = NEW_APPLICATION.*
RESTART APP SERVER AS NEW APPLICATION.*
IF INCOMING = NEW_MODEL.*;
STOP SERVER
MODEL = NEW_MODEL.*
RESTART SERVER
SENSOR
APACHE MINIFI
CLOUD

Figure 18: Overview of logic within system components that make up the task deployment scheme on the Edge container. Solid line represents sensor data moving through the system. Dotted line represents control information from the Cloud moving through the system. Minifi is responsible for the dataflow through the system.

3.4.2 Central Container System Components:

User Configurations: GUI or service UI that allows the developer to pass control information to the Edge devices.

Cloud Processing Server: Listens for incoming data from the Edge container in situations where task offloading occurs due to limitations or restraints on the Edge device.

Cloud Storage: A large storage unit that holds all data ingested from the Edge containers. Further analysis takes place and future models are built from this stored data.

Figure 19 presents an overview of components on both Edge and Central container. Solid black lines represent data flow, dotted lines represent control information flow whereas red lines represent data transmitted as a result of offloading.

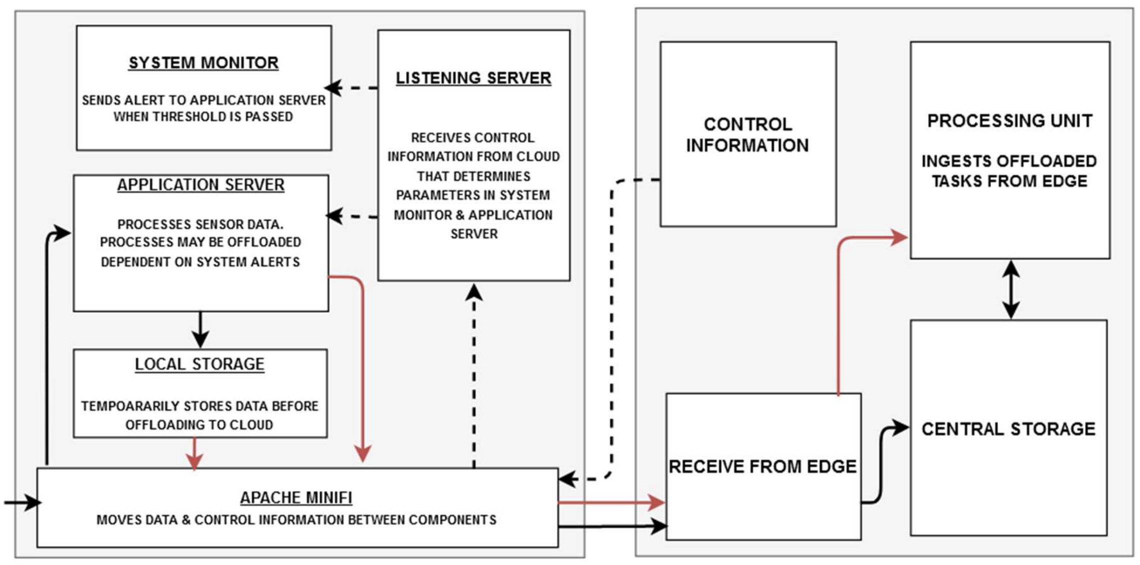

SYSTEM MONITOR
SENDS ALERT TO APPLICATION SERVER WHEN THRESHOLD IS PASSED
LISTENING SERVER
RECEIVES CONTROL INFORMATION FROM CLOUD THAT DETERMINES PARAMETERS IN SYSTEM MONITOR & APPLICATION SERVER
APPLICATION SERVER
PROCESSES SENSOR DATA. PROCESSES MAY BE OFFLOADED DEPENDENT ON SYSTEM ALERTS
LOCAL STORAGE
TEMPOARARILY STORES DATA BEFORE OFFLOADING TO CLOUD
APACHE MINIFI
MOVES DATA & CONTROL INFORMATION BETWEEN COMPONENTS
CONTROL INFORMATION
PROCESSING UNIT
INGESTS OFFLOADED TASKS FROM EDGE
RECEIVE FROM EDGE
CENTRAL STORAGE

Figure 19: Overview of system components. Solid black lines represent data flow, dotted lines represent control information flow whereas red lines represent data transmitted as a result of offloading

3.5 DAGLADS Implementation & Evaluation

To evaluate DAGLADS, we consider two implementations. Implementation I evaluates DAGLADS against the current state of the art centralized approach. Here, we aim to show how the powerful processing technologies such as the centralized Apache Storm, can be emulated on less powerful edge devices. Apache Nifi/MiNifi are primarily data moving technologies with minimal processing capabilities. However, here we combine Nifi with custom applications on the edge devices, enhancing Edge processing capabilities that are in line with the Cloud approach. Implementation II further enhances Edge processing capabilities through combining Nifi with learning algorithms for local processing, and custom techniques for transmitting data from Edge to Cloud efficiently. For our implementations, we use Raspberry Pi's as the Edge devices and a HP Laptop as the Cloud. The specifications of the Pi are as follows: 1 GB of Ram and a CPU; 4× ARM

Cortex-A53, 1.2GHz. The specifications of the HP laptop are 16GB of Ram and a CPU; Intel Core i5, 8th generation.

3.5.1 Implementation I

First, a comparison is made against a well-known centralized approach demonstrated by Hortonworks. A streaming analytics use case for a fleet of trucks as specified in [122] is considered. This evaluation considers driver performance consisting of average speeds and unsafe event notifications. Trucks generate millions of events for any given route. Normal events include vehicle starting, vehicle stopping etc. Violation events include speeding, excessive acceleration, excessive breaking and unsafe tail distance. The Business Requirement of the HDF demo is to stream the trucking events in, filter on violations and do real-time alerting when "lots" of erratic behaviour is detected for a given driver over a short period.

In the HDF trucking application, a data simulator creates the data. Apache Nifi, located in the Cloud ingests all data from the trucks and separates the incoming data into two dataflows. As shown in Figure 20, the first dataflow, "truck_geo_events", extracts the lines of data featuring the truck events. The second dataflow, "truck_speed_events", extracts the lines of data featuring the speed of the truck. These dataflows are forwarded to Kafka Topics and to Storm Spouts. Storm bolts calculate the average speeds of the drivers over a specified timeframe and also filters violations and performs real-time analysis, detecting erratic behaviour for a driver over a short period. If a driver creates five violations in a three-minute window, an alert is sent directly to the fleet manager.

The HDF application is further enhanced with a collaboration of Spark MLlib, which transforms HDF from a streaming application to a prediction application. This is achieved by building a model from the trucking company's datasets. Information such as historical truck events are enriched with weather and payroll context, which provides a model for prediction of future driver violations. A GUI is provided, Figure 21 , showing the driver routes and events in real time.

The goal of this implementation is to compare DAGLADs processing capabilities against the more powerful centralized approach, with the aim of maintaining computational accuracy while minimizing data transmission to the Cloud. In Implementation 1.1, we extend the processing from the centralized HDF approach to the trucks. Apache MiNifi, combined with custom applications, is used on the trucks to perform processing and filtering before forwarding data to the main Nifi server, which incorporates custom applications to enrich driver violations with weather context in real-time. Implementation 1.2 evaluates an enhanced architecture in which all processing, filtering and data enrichment occur on the trucks.

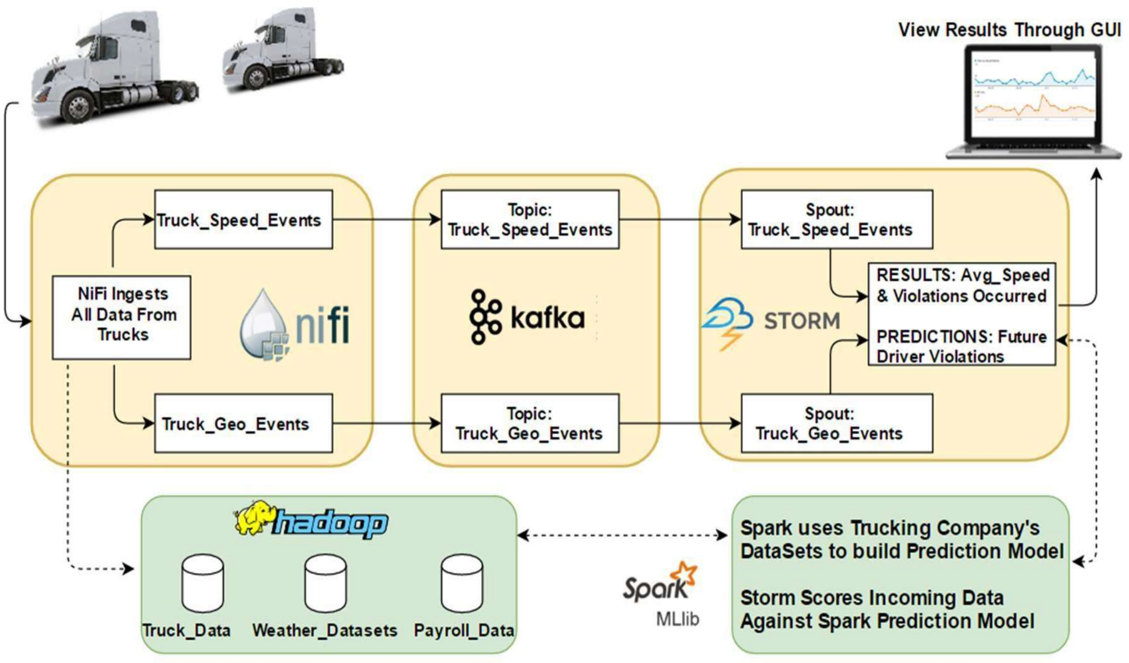
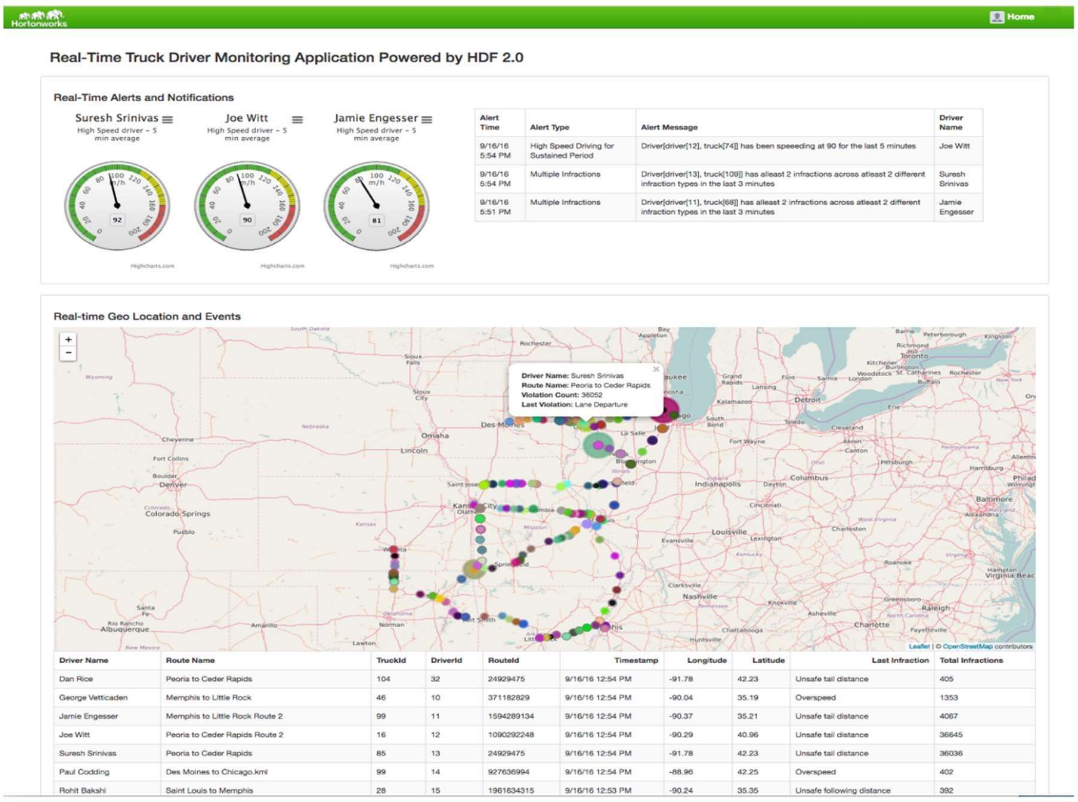

View Results Through GUI
NiFi Ingests All Data From Trucks
Truck_Speed_Events
Truck_Geo_Events
nifi
Topic: Truck_Speed_Events
kafka
Topic: Truck_Geo_Events
Spout: Truck_Speed_Events
STORM
RESULTS: Avg_Speed & Violations Occurred
PREDICTIONS: Future Driver Violations
Spout: Truck_Geo_Events
hadoop
Truck_Data
Weather_Datasets
Payroll_Data
Spark MLlib
Spark uses Trucking Company's DataSets to build Prediction Model
Storm Scores Incoming Data Against Spark Prediction Model

Figure 20: HDF with Nifi ingesting data and passing to storm for processing


Hortonworks
Home
Suresh Srinivas
High Speed driver - 5 min average
92
Joe Witt
High Speed driver - 5 min average
90
Jamie Engesser
High Speed driver - 5 min average
81
Highcharts.com
Driver Name: Suresh Srinivas
Route Name: Peoria to Ceder Rapids
Violation Count: 36052
Last Violation: Lane Departure
Leaflet | © OpenStreetMap contributors

Real-Time Truck Driver Monitoring Application Powered by HDF 2.0

Real-Time Alerts and Notifications

Alert Time	Alert Type	Alert Message	Driver Name
9/16/16 5:54 PM	High Speed Driving for Sustained Period	Driver[driver[12], truck[74]] has been speeeding at 90 for the last 5 minutes	Joe Witt
9/16/16 5:54 PM	Multiple Infractions	Driver[driver[13], truck[109]] has alleast 2 infractions across atleast 2 different infraction types in the last 3 minutes	Suresh Srinivas
9/16/16 5:51 PM	Multiple Infractions	Driver[driver[11], truck[68]] has alleast 2 infractions across atleast 2 different infraction types in the last 3 minutes	Jamie Engesser

Real-time Geo Location and Events

Driver Name	Route Name	TruckId	DriverId	RouteId	Timestamp	Longitude	Latitude	Last Infraction	Total Infractions
Dan Rice	Peoria to Ceder Rapids	104	32	24929475	9/16/16 12:54 PM	-91.78	42.23	Unsafe tail distance	405
George Vetticaden	Memphis to Little Rock	46	10	371182829	9/16/16 12:54 PM	-90.04	35.19	Overspeed	1353
Jamie Engesser	Memphis to Little Rock Route 2	99	11	1594289134	9/16/16 12:54 PM	-90.37	35.21	Unsafe tail distance	4067
Joe Witt	Peoria to Ceder Rapids Route 2	16	12	1090292248	9/16/16 12:54 PM	-90.29	40.96	Unsafe tail distance	36645
Suresh Srinivas	Peoria to Ceder Rapids	85	13	24929475	9/16/16 12:54 PM	-91.78	42.23	Unsafe tail distance	36036
Paul Codding	Des Moines to Chicago.kml	99	14	927636994	9/16/16 12:54 PM	-88.96	42.25	Overspeed	402
Rohit Bakshi	Saint Louis to Memphis	28	15	1961634315	9/16/16 12:53 PM	-90.24	35.35	Unsafe following distance	392

Figure 21: Real-time GUI showing driver routes and behaviour

3.5.2 Implementation II

Implementation I introduced the capabilities of FBP and the benefits of processing locally. However, it is entirely unidirectional, and not a true representative of the proposed architecture. Implementation II is a representation of this authors published papers [120] and [121] and illustrates a distributed configuration management protocol, which coordinates processing between the Edge and Central Container. A vehicle use case that predicts driver alertness is evaluated. The dataset used for this work was initially proposed in a Kaggle competition called “Stay Alert! The Ford Challenge” [123]. The objective was to design a classifier that detects whether the driver is alert or not, employing data acquired from over 100 participants while driving. The dataset consists of 30 features. Eight of these features are Physiological and are represented with a P, (P1, P2, P3 etc.). 11 are Environmental, represented with E. 11 are Vehicular features, and represented with V. For each observation, an output “IsAlert” is labelled with 1 indicating that the driver is alert or 0 if not alert.

In this implementation, advanced data mining libraries are installed on the Edge device. Driver drowsiness is predicted locally without the need to transmit to the Central container. This is advantageous as the risk of network bottleneck and high latency is no longer an issue, considering alerting a driver may be considered mission critical. Results are then stored locally. A system is also created that will transmit a summarization of data every minute to the Cloud. One of the drawbacks for many Edge devices is the lack of local storage. To address this, a system is created that compresses blocks of data every ten minutes and transmits to the Central server, reducing the local disk usage space while transmitting all data at a significantly lower rate.

In implementation II, TPOT is used on the Central container to build and optimize the algorithm most suited for the dataset. AutoML algorithms are not as simple as fitting one model on the dataset; they are considering multiple machine learning algorithms with multiple pre-processing steps. With the default settings, TPOT will evaluate 10,000 pipeline configurations before finishing, meaning it will iterate through 10,000 combinations of hyper parameters to find the best performing model. The user can alter these parameters to perform a quicker search if necessary. For this use case, we evaluated 500 pipeline configurations, which took 2 hours. TPOT is incorporated into the architecture to automatically build the most suitable models, where Nifi and MiNifi are configured to automatically distribute the TPOT model to the Edge devices.

This section will demonstrate the advantages of including TPOT to automatically build the models before distribution. No feature engineering was performed on the data before executing TPOT. The training dataset was split 70-30, building the model on 30% of the data and validating the model on the remaining 70%. After running TPOT for two hours on the training data, it provided a model with the most optimized hyper parameters. The chosen model was an ExtraTreeClassifier. As a comparison, five other algorithms were tested, with just their default settings. Note, one of the algorithms we tested was an ExtraTreeClassifier with its default settings. This aims to show the improvement optimizing hyper parameters via TPOT can make.

Table 5 represents the individual model accuracies that were tested. As shown, the automated model built by TPOT scored highest, enhancing the accuracy by 0.3% and the auc by 1.1% for this particular dataset. Overall, TPOTs inclusion to the architecture is justified due to its improved results and automated nature.

Table 5: Model Scores on Validation Dataset

Algorithm	Accuracy (%)	ROC_AUC (%)
Logistic Regression	78.7	77.6
KNearest Neighbour	83.2	82.3
RandomForestTree Classifier	97.6	97.4
DecisionTree	96.2	96.1
ExtraTreeClassifier	97.8	96.6
TPOT Model	98.1	97.7

3.6 DAGLADS Results

The section describes the results obtained from the implementations of our proposed architecture. Much of this section is taken from already published work from the author and will be displayed in a format that shows the evolution of the architecture. Implementation I will provide an introductory approach to Flow Based Programming and its Edge processing capabilities and has been published in [119]. Implementation II is the result of publications [120], [121] and aims to show the benefits of adding advanced analytical libraries to the Edge containers, which will provide a final instalment of implementations that include a computational collaboration between Edge and Central container via task offloading from the Edge to the Cloud. Apache Nifi and its sub project MiNifi are the FBP applications used in the following scenarios. It must be noted, Nifi/MiNifi are primarily data streaming technologies with no processing capabilities. However, combined with the author's custom software, the goal is to emulate powerful centralized processing technologies on Edge devices.

3.6.1 Implementation 1.1: Local Processing

MiNifi is installed on Raspberry Pi's representing the trucks. The goal is to emulate the processing achieved by Storm in the centralized HDF approach. Figure 22 shows the dataflows created through MiNifi. Dataflow 1 calculates the average speed. Dataflow 2 filters unsafe events. Dataflow 3 determines if five unsafe events have occurred within three minutes. If true, an email is sent directly to the fleet manager. All data is saved locally which can be uploaded in batch at a later stage. The main Nifi server ingests the data from the Edge devices for further processing.

To separate the incoming sensor data into individual dataflows, a RouteText processor is configured with regular expressions that forward truck_speed_events and truck_geo_events to separate dataflows. Truck_geo_events is subsequently split into two further dataflows. The Speed_events data, (dataflow 1), consists of a MergeContent processor that merges single flowfiles into a user specified amount. This is useful when performing computations that require a specified timeframe of data or a certain amount of data. An UpdateAttribute processor is configured to give the merged content a filename. This allows us to create a sliding window that provides an average speed over five seconds. Once the processing has been completed, the results can be stored locally or forwarded to the main Nifi server.

Dataflow 2 uses a RouteText processor configured to filter out all "unsafe" events, which are then forwarded to the main Nifi server. In the HDF use case, Storm sends an alert to the fleet manager whenever a driver creates five unsafe events in a three-minute time window. Dataflow 3 emulates this action using a RouteText processor to filter out unsafe events. This data is merged into a file containing three-minutes of data. A custom program acting a sliding window, calculates if five violations have occurred in the specified timeframe. If true, a PutEmail processor will alert the fleet manager immediately with a file containing a list of violations occurred.

The Nifi server ingests and separates incoming data from the Edge devices, Figure 23. The average speed data is extracted and can be forwarded to a dashboard for further analysing, or forwarded to storage. Unsafe events are also extracted. A dataflow on the main Nifi server enriches all unsafe events with real-time weather attributes. This dataflow consists of a number of processors that extract the latitude and longitude attributes before sending them to a weather API using an InvokeHTTP processor, Figure 25, and receives an immediate weather response in JSON format. An EvaluateJSONpath processor parses the JSON file for weather in relation to wind, rain and fog. These new weather attributes are then appended to the original Geo_events flowfile. A conversion is executed on the incoming data to translate attributes including weather and events into binary, preparing it for the Spark prediction model, as shown in Figure 24. This is achieved through a custom python application.

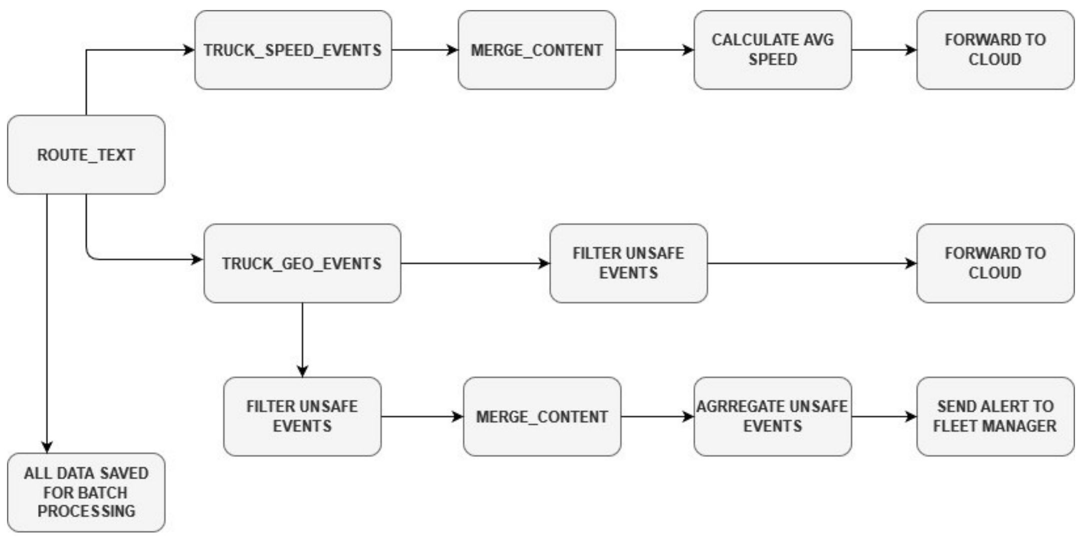


Figure 22: Dataflows created on Raspberry Pi, emulating local processing on truck

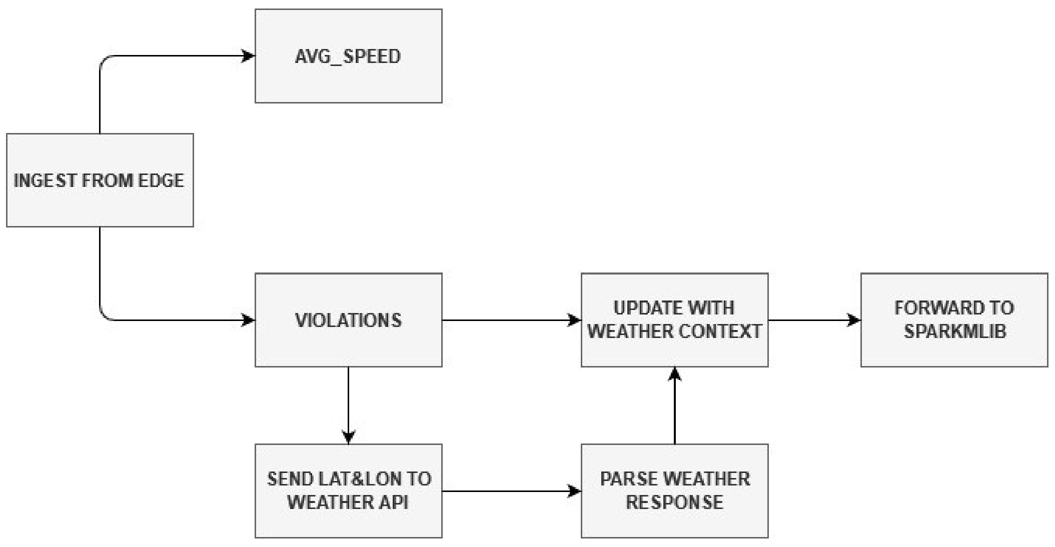
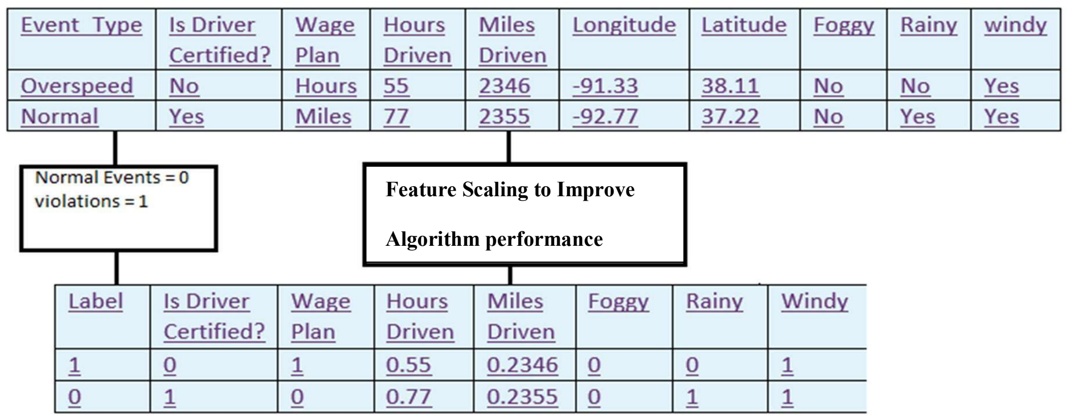


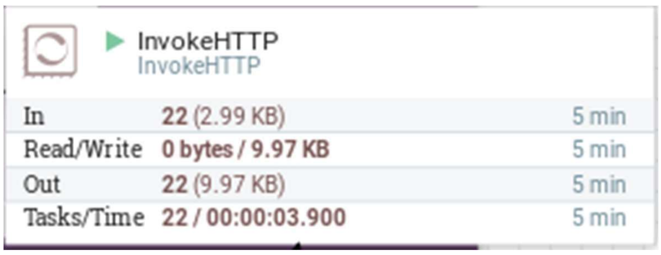
Figure 23: Dataflows created on Apache Nifi server emulating Cloud server



Event Type	Is Driver Certified?	Wage Plan	Hours Driven	Miles Driven	Longitude	Latitude	Foggy	Rainy	windy
Overspeed	No	Hours	55	2346	-91.33	38.11	No	No	Yes
Normal	Yes	Miles	77	2355	-92.77	37.22	No	Yes	Yes

Label	Is Driver Certified?	Wage Plan	Hours Driven	Miles Driven	Foggy	Rainy	Windy
1	0	1	0.55	0.2346	0	0	1
0	1	0	0.77	0.2355	0	1	1

Figure 24: Enriching and formatting truck events for Spark Model


InvokeHTTP
InvokeHTTP

```
In          22 (2.99 KB)              5 min
Read/Write  0 bytes / 9.97 KB         5 min
Out         22 (9.97 KB)              5 min
Tasks/Time  22 / 00:00:03.900         5 min
```

Figure 25: Information passing through InvokeHTTP processor over 5 minute period

Figure 25 shows an example of the information each processor provides over a five minute period. This includes how much data is ingested by the processor (In), and how much leaves the processor (Out). In this example, the InvokeHTTP processor is configured to post incoming latitude and longitude features to a weather API, and retrieve current weather status from that area. As shown, the processor ingested 2.99KB of data, enriched it with weather context from the weather API and passed 9.97KB of data to the following processor. This occurred 22 times over the five-minute period.

The simulation is configured to create data for both 1 driver and 23 drivers. The quantity of data produced is controlled by increasing the granularity of data production from 500ms, 250ms to 100ms. Table 6 illustrates the quantity of data transmitted from the Edge device in the single driver evaluation.

Table 6: Quantity of Data Transmitted – Single Driver (Without Weather Context)

Data Interval (ms)	**Centralized Approach -All Data Transmitted to Cloud (KB)**	**Edge Device Processing (KB)**	**Data Reduction**
100	625.76 KB	10.43 KB	98.33%
250	297.82 KB	8.72 KB	97.07%
500	141.48 KB	8.66 KB	93.88%

Figure 26 graphically represents the data from Table 6. It illustrates that the architectural implementation reduces the quantity of data transmitted for centralized processing by up to 98%, while maintaining computational accuracy.

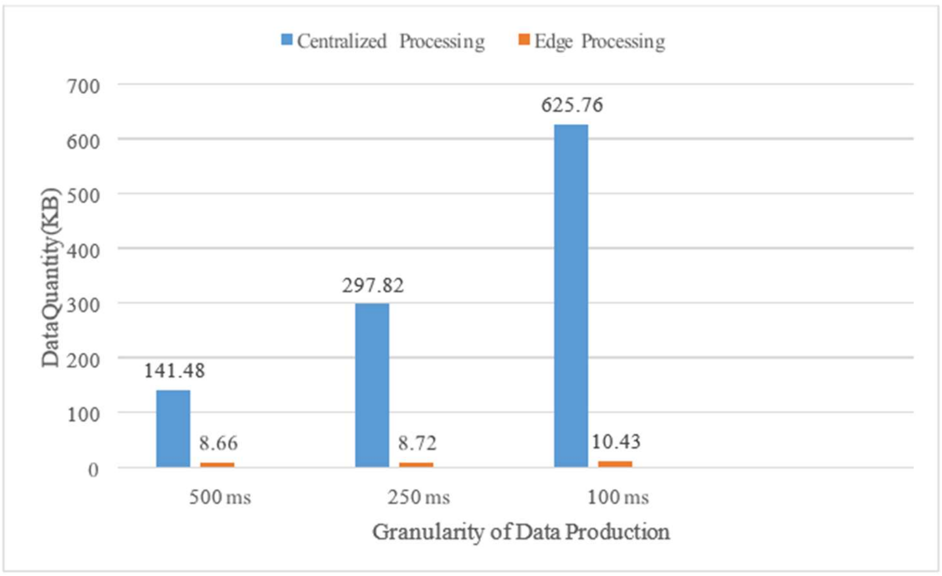


Figure 26: Quantity of Data Transmitted – Single Driver (Without Weather Context)

Table 7 illustrates the quantity of data transmitted from an Edge devices for a 23-driver evaluation when the effect of weather context was not considered. Figure 27 graphically represents the results of the same simulation.

Table 7: Quantity of Data Transmitted – 23 Drivers (Without Weather Context)

Data Intervals (ms)	**Centralized Approach (KB)**	**Edge Processing (KB)**	**Data Reduction**
100	13240	344.7	97.4%
250	6140	162.66	97.35%
500	3240	88.95	97.25%

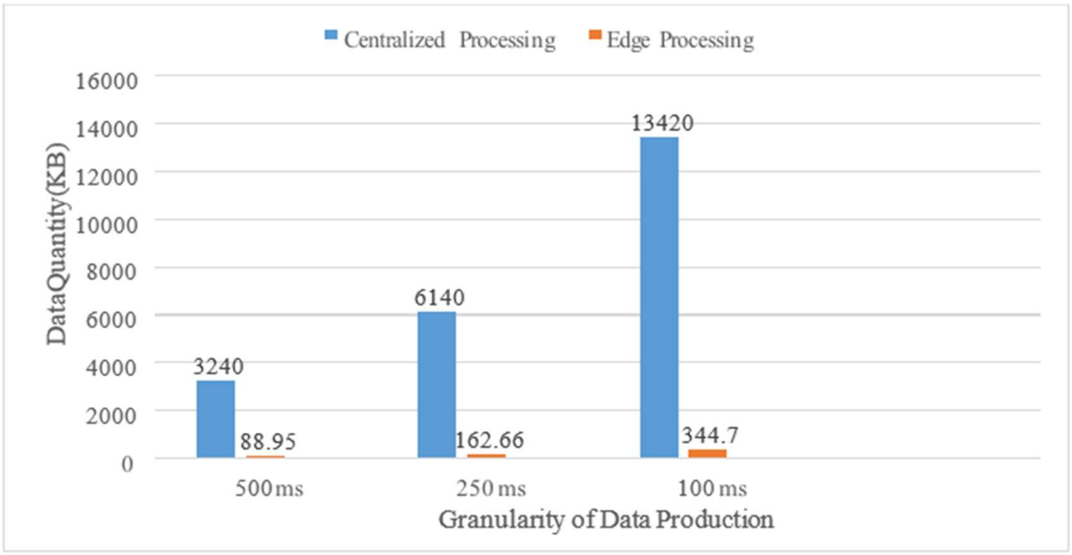


Figure 27: Quantity of Data Transmitted – 23 Drivers (Without Weather Context)

Table 6 and Table 7 illustrate that from both a single driver and 23 drivers evaluation, as the velocity of data increased a slight increase in performance was experienced. In the single driver scenario there was a 93.88% reduction in data transmitted when data was processed on the Edge device at 500ms granularity. When the data production rate was increased by a factor of 5, with data produced every 100ms the data reduction by processing on the Edge device increased to 98.33%. The improved performance results from merged content when calculating the average speed. When data was created at higher velocity, more content needed to be merged to aggregate 5 seconds of data, yet still resulting in one output.

3.6.2 Implementation 1.2 – Data Enrichment on the Edge

In this section, we describe the architectural implementation where processing, filtering and data enrichment of weather context occur on the Edge devices. Figure 28 illustrates the three dataflows created on the Edge device for this scenario.

Dataflow 1 calculates the average speed.

Dataflow 2 enriches each unsafe event with weather attributes, and prepares it for the SparkML prediction model. Dataflow 3 is identical to dataflow 3 in scenario 1.

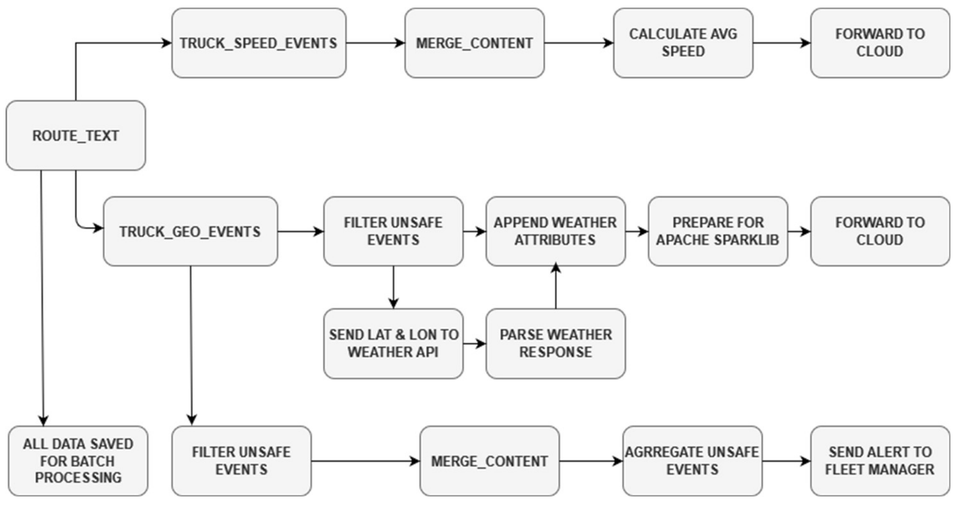


Figure 28: Architecture 2 – Dataflows on Edge for Processing, Enriching, and Filtering

Figure 29 is an actual image of the dataflows created in Nifi. As shown, sections of the dataflow can be named and coloured. This is advantageous in keeping track of different sections of a dataflow.

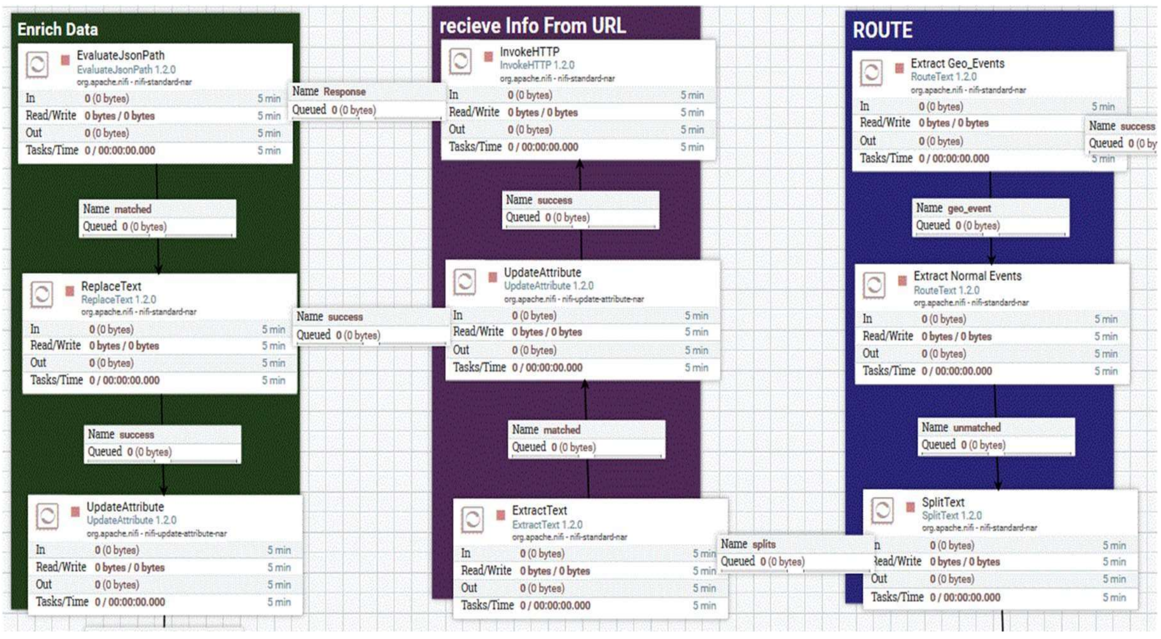


Figure 29: Actual image of dataflow where enrichment occurs.

In this section, we compare the performance of centralized processing and Edge processing considering the effect of weather conditions. Table 8 illustrates the quantity of data transmitted from the Edge device in the single driver evaluation with weather context.

Table 8: Quantity of Data Transmitted – 1 Driver (With Weather Context)

Data Granularity (ms)	Central Approach (KB)	Edge Approach (KB)	Data Reduction
100	626.43	37.97	93.94%
250	288.57	27.18	90.58%
500	144.58	17.22	88.09%

Figure 30 is a graphical representation of the results. It illustrates that our enhanced architectural implementation reduces the quantity of data required to be transmitted for centralized processing by up to 94% when the context of weather conditions is considered.

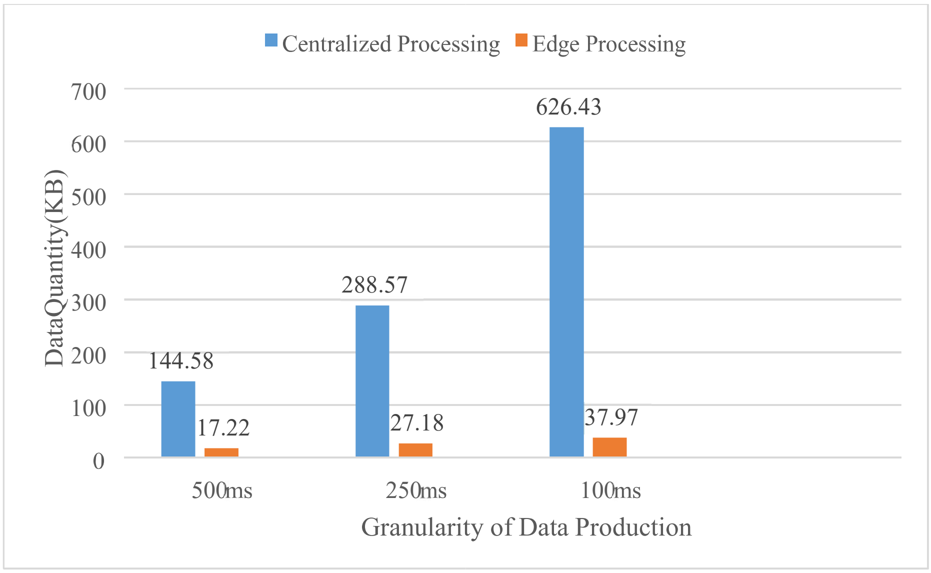


Figure 30: Quantity of Data Transmitted – 1 Driver (With Weather Context)

Table 9 illustrates the quantity of data transmitted from the Edge device for a 23-driver evaluation when the effect of weather was considered. Figure 31 graphically represents the data.

Table 9: Quantity of Data Transmitted – 23 Drivers (With Weather Context)

Data Intervals (ms)	Central Approach (KB)	Edge Approach (KB)	Data Reduction
100	13430	2659.6	80.2%
250	6140	1153.66	81.21%
500	3230	620.26	80.8%

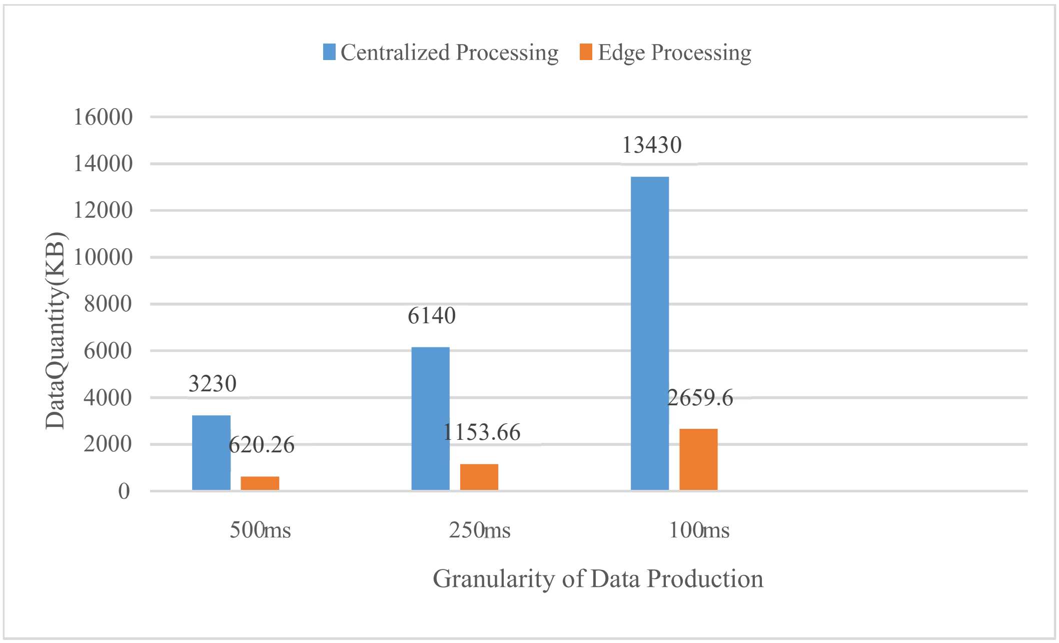


Figure 31: Quantity of Data Transmitted – 23 Drivers (With Weather Context)

Table 8 and Table 9 illustrate that when the context of weather is included; the Edge processing approach still has good performance in comparison to a centralized processing approach when considering data transmission and computational accuracy. The results do however illustrate that when weather context is considered, dangerous drivers create more unsafe events. These unsafe events generate multiple weather requests, which are processed on the Edge device.

3.6.3 Implementation 2.1 – Advanced Local Processing

Here we focus on the performance of adding advanced analytical libraries to the Edge containers. As discussed, Implementation II focuses on predicting driver drowsiness, and demonstrates computational collaboration between Edge and Central container via task offloading from the Edge to the Cloud. MiNifi, python and Sci-kit libraries are installed on a Raspberry Pi representing the Connected Vehicle. A dataflow consisting of multiple processors are installed via MiNifi. The test dataset is placed in the Pi. A SplitText processor is configured to ingest the data one line at a time from the test dataset, followed by a ControlRate processor configured to set the rate at which each flowfile travels through the dataflow. Here, it is set at one flowfile every 2 seconds, emulating the vehicle creating data in real time. An UpdateAttribute processor assigns each feature within the flowfile an attribute name. This allows the attributes to be split and routed separately if necessary. Figure 34 shows a representation of the architecture.

The next step in the dataflow is to duplicate the incoming flowfile and route it separately to the prediction model in the application server, and the summarization process. This is achieved using a RouteText processor as shown in Figure 32. The processor is configured with regular expressions; flowfiles matching the regular expressions can be forwarded one direction, with the unmatching flowfile forwarded elsewhere, or dropped. In this scenario, the flowfile is duplicated and forwarded to two separate dataflows.

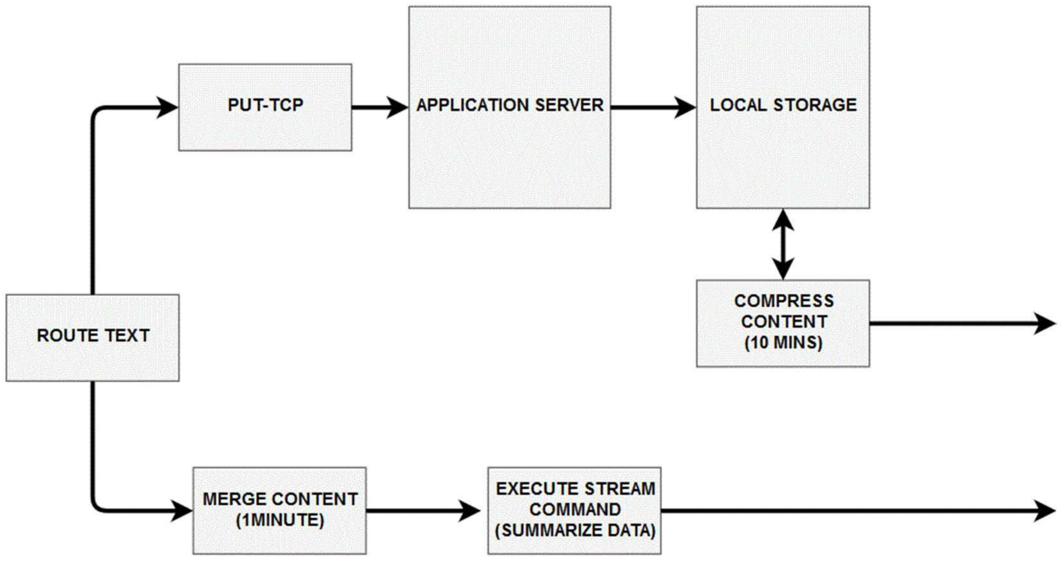
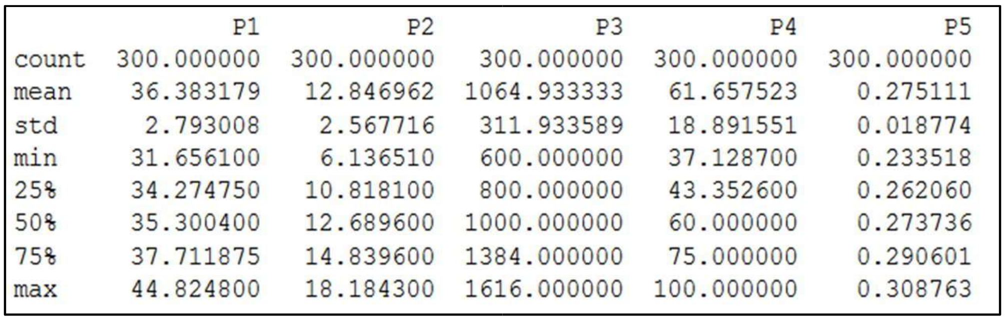


Figure 32: Dataflows on Edge Container

The first dataflow forwards the flowfiles to an application server via PutTCP processor. Here, a custom python application, which uses the sci-kit library, makes a prediction on the incoming data against the prediction model. The output is appended to the flowfile, and stored locally for future model building. If driver drowsiness is predicted, an alert can be sent to the driver’s phone and/or the fleet manager’s office.

The second dataflow forwards the flowfiles to a MergeContent Processor. In this scenario, one minute of data is merged. An ExecuteStreamCommand processor calls another custom python application that summarizes the merged block of data. Figure 33 shows the output of this process. Transmitting summarized data can be beneficial as sending data frequently is resource heavy. A separate dataflow merges data together over a longer period, in this example, 10-minute blocks, before compression. Transmitting compressed data greatly reduces data transmission. The Nifi server ingests the compressed data from the Edge devices, which is then decompressed and stored for further analysing.



```
               P1          P2           P3          P4          P5
count  300.000000  300.000000   300.000000  300.000000  300.000000
mean    36.383179   12.846962  1064.933333   61.657523    0.275111
std      2.793008    2.567716   311.933589   18.891551    0.018774
min     31.656100    6.136510   600.000000   37.128700    0.233518
25%     34.274750   10.818100   800.000000   43.352600    0.262060
50%     35.300400   12.689600  1000.000000   60.000000    0.273736
75%     37.711875   14.839600  1384.000000   75.000000    0.290601
max     44.824800   18.184300  1616.000000  100.000000    0.308763
```

Figure 33: Summarized data over a 5-minute period (count 300 seconds)

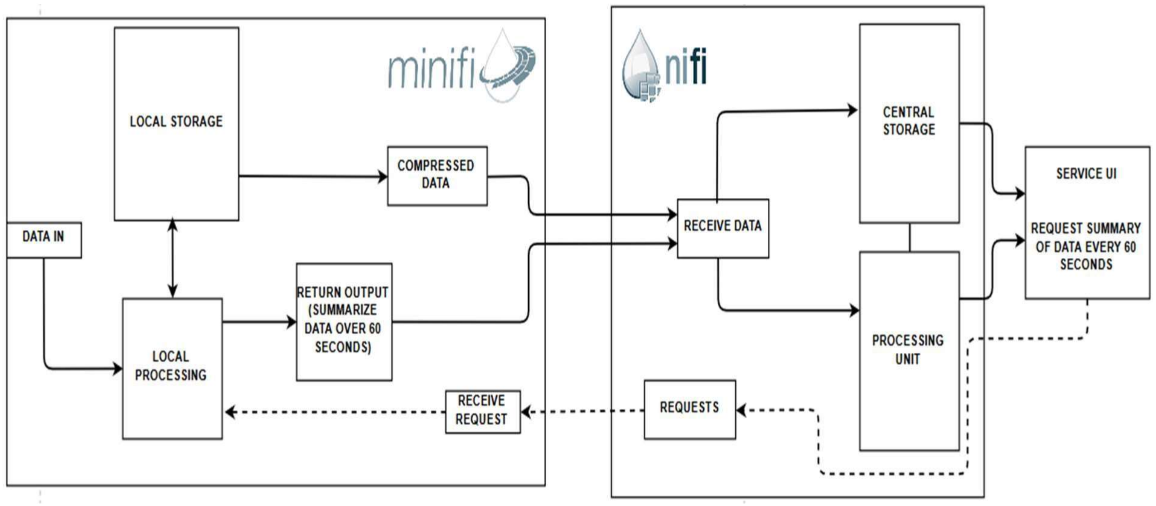


Figure 34: Service UI requesting summary of data every 60 seconds

The reduction of data transmission was recorded in multiple scenarios. This was achieved by increasing the control rate at which data passed through the Edge device. The quantity of data produced is controlled by increasing the granularity of data production from 2 seconds, 1 second, and 500 ms. Table 10 represents the data reduction over a 5-minute period. Full data transmission is compared against the data summary and compressed data columns combined. As pointed out in previous implementations, as velocity of data increased a significant increase in data reduction occurred. The improved performance resulted from merging the content. When data was created at a higher velocity, more content was merged in the specified period. However, summarized output remained the same.

Table 10: Comparison of full data transmission against summarized and compressed data

Data Intervals (ms)	**Full Data Transmission**	**Compressed Summary (1 minute intervals)**	**Full Compressed data (10 Min intervals)**	**Total Data Reduction**
2000	20.5KB	1.44 KB	2.95 KB	78.59%
1000	39.5 KB	1.46 KB	6.6 KB	79.6%
500	79.9 KB	1.48 KB	12.5 KB	82.5 %

3.6.4 Implementation 2.2: Collaborative Processing / Task Offloading

As discussed, DAGLADS has the capability of offloading tasks from the Edge container to the Cloud. Guaranteeing low-latency applications and services to the end users will be fundamental for the Edge/Fog-Computing paradigm. To provide this type of service, processing as close to the source as possible will be necessary. However, in certain scenarios, this may not be achievable due to a number of circumstances. In such a case, offloading certain tasks to the Cloud may free up the already limited resources on the Edge containers. Offloading may be determined by multiple factors, including, but not limited to, battery power, latency, and local environmental factors.

It will be common for Edge containers to be battery powered. In such a scenario, to preserve resources it may prove beneficial to process the most time critical data locally and offload less time critical tasks when battery power is falling below a certain threshold. However, in this scenario, computation offloading based on battery level was not necessary. Instead, we set thresholds based on CPU usage. Three data flows are created on the Edge container as depicted in Figure 35. Dataflow 3 is considered the most time critical data that must be processed locally at all times.

In this scenario, features that determines driver drowsiness is presumed critical. Dataflow 1 and 2 represent data that can be processed locally, but not with such demand on low latency. These dataflows represent features such as latitude, longitude, speed, Driver_ID, Truck_ID etc. When CPU usage rises above a specified threshold, the computations that occur on dataflow 1 and 2 are offloaded to the Cloud. The Cloud ingests the data from 1 and 2 and processes centrally. Dataflow 1 and 2 consist of an ExecuteStreamCommand processor that calls a python script to perform local analysis. Dataflow 3 is routed to the local application server and is constantly processed locally.

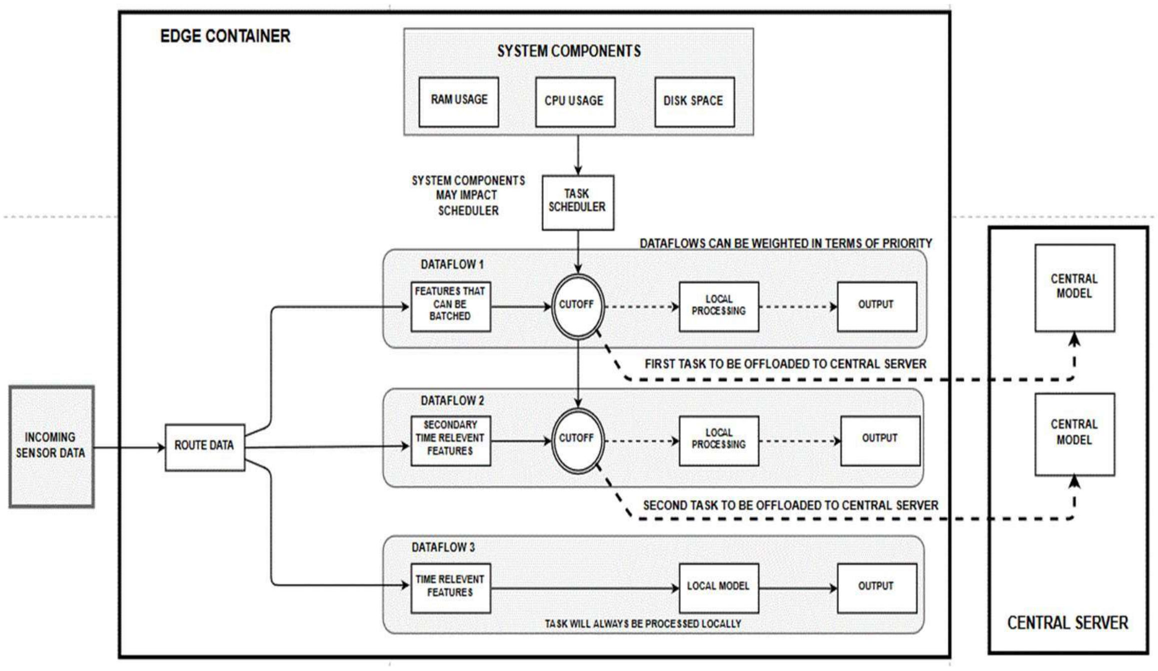

EDGE CONTAINER
SYSTEM COMPONENTS
RAM USAGE
CPU USAGE
DISK SPACE
SYSTEM COMPONENTS MAY IMPACT SCHEDULER
TASK SCHEDULER
DATAFLOWS CAN BE WEIGHTED IN TERMS OF PRIORITY
DATAFLOW 1
FEATURES THAT CAN BE BATCHED
CUTOFF
LOCAL PROCESSING
OUTPUT
FIRST TASK TO BE OFFLOADED TO CENTRAL SERVER
DATAFLOW 2
SECONDARY TIME RELEVENT FEATURES
CUTOFF
LOCAL PROCESSING
OUTPUT
SECOND TASK TO BE OFFLOADED TO CENTRAL SERVER
INCOMING SENSOR DATA
ROUTE DATA
DATAFLOW 3
TIME RELEVENT FEATURES
LOCAL MODEL
OUTPUT
TASK WILL ALWAYS BE PROCESSED LOCALLY
CENTRAL MODEL
CENTRAL MODEL
CENTRAL SERVER

Figure 35: Dataflow 3 is time critical and is processed locally at all times. Dataflow 1&2 may be offloaded

A separate system monitors CPU usage at regular intervals. For this scenario, a threshold of 75% CPU usage was set. A bash programming application, which represents a task scheduler, is configured to trigger an alert if CPU usage rises above the threshold for over 30 seconds. If true, dataflow 1 will cease processing locally and direct the incoming flowfiles directly to the Cloud container. After this action, if CPU remains above 75%, dataflow 2 is then offloaded, (This may be true when other tasks such as system updates etc. are occurring). This can be reversed when CPU returns below the threshold. Tasks/computations on the dataflows may be modified in real time to keep the CPU usage below the threshold.

System monitoring and resource management may influence decision making in terms of task resource allocation on Edge containers by providing useful information such as workload and energy usage. The efficiency of advanced embedded systems will play a major role in IoT. As previously mentioned, the Edge processing for this work was carried out on a raspberry Pi. The specifications of the Pi are as follows: 1 GB of Ram and a CPU; 4× ARM Cortex-A53, 1.2GHz.

Three python scripts were running simultaneously in this scenario (one on each dataflow). Each python script was configured to ingest one flowfile at a time, at intervals of 500 ms, and score the data off a local model. With Apache MiNifi and the three python scripts running, CPU usage was on average 77%.

In this scenario, dataflow 1 was offloaded, which lowered the overall CPU usage to 51.7%. To keep CPU usage below the threshold, a new python application was implemented to slightly adjust the manner of which data was processed in dataflow 1. For example, by modifying the application to perform a minor process, such as feature selection instead of prediction, before transmitting to a Central container for further analysing, all three dataflows could be run concurrently, and below the threshold. The newly modified dataflow gave a total average CPU usage of 66%. Figure 36 shows the Edge containers CPU usage before and after the modification, compared to the threshold.

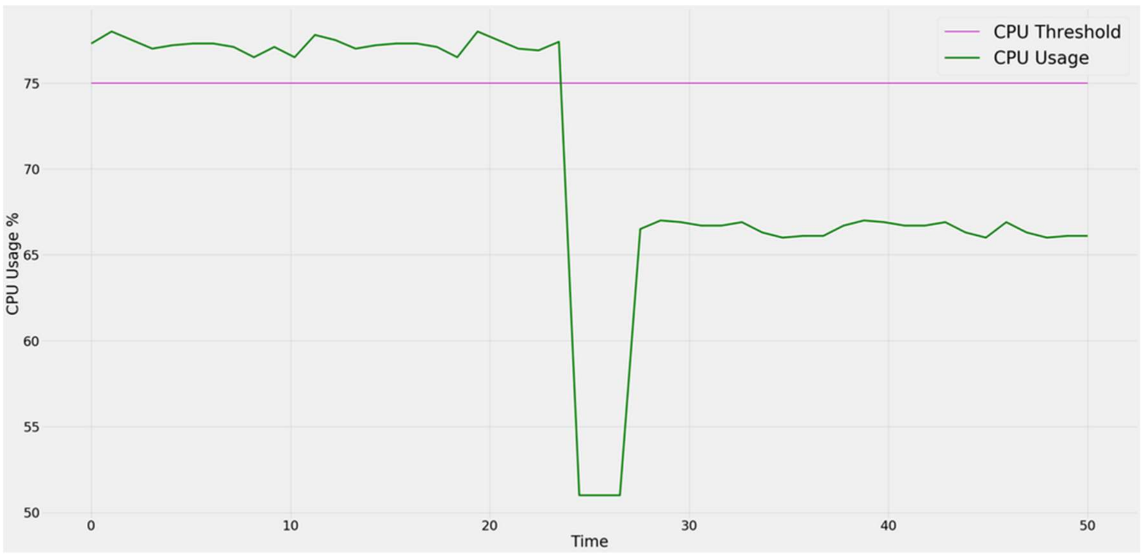


Figure 36: Drop in CPU usage occurs when dataflow 1 is offloaded

3.7 Conclusion

This section discussed the individual architectural and system components necessary to meet the following requirements and fundamentals that must be considered when defining a DAGLADS architecture.

Latency Sensitivity – The proposed architecture offers the end users low-latency guaranteed applications and services, due to the advanced local processing capabilities via the application server provided on the Edge container, as shown in Implementation II.

Generality - With the inclusion of Apache MiNifi, and its wealth of processors, the architecture is capable of handling a diversity of data types, and data generation rates.

Dynamism and Efficiency– The inclusion of a monitoring system allows for the self-adaption of the IoT application on the Edge container. The monitoring system also provides a set of rules that efficiently utilize resources on the Edge container

Programmability – Developers can deploy and modify the applications remotely through the service UI on the Central container.

System evaluation and results of DAGLADs were presented through multiple implementations. Implementation I showed the role in which FBP can play in an IoT scenario. Although Apache Nifi and MiNifi are primarily data streaming technologies, when combined with custom applications they can perform advanced processing capabilities. Due to their wealth of processors that can be configured to suit almost any IoT scenario, and the bi-directional capabilities, confirm FBP based technologies are a valuable addition to an IoT architecture. Implementation II focused on advanced Edge analytics demonstrating data summarization and task offloading, showing a collaboration between the Edge and Central container. This section provided sufficient results that showed the importance of Edge analytics in the success of IoT. Significant reduction in data transmission lowers the risk of network bottlenecks and also lowers costs in relation to bandwidth and storage for businesses.

The contribution of DAGLADs can be viewed as follows.

1) An IoT platform that combines FBP and the authors' custom software to emulate the processing capabilities of the current state of the art centralized approach.
2) A bi-directional IoT platform in which developers can seamlessly distribute control information and logic to Edge devices.

4 In-Vehicle Data Capturing

4.1 System Design & Methodology

Vehicle and driving behaviour information is instrumental when developing traffic applications and accident prevention systems throughout a Smart City. Aggressive accelerations and decelerations for example, can determine the necessity of speed bumps, or lower speed limits, in certain areas. High RPM, idle time, and CO_2 emissions can give insight into unnecessary length of time spent at traffic junctions. Stop/start events mean heavy traffic or roadblocks. Therefore, utilizing the most accurate data capturing technique is essential.

This section focuses on the development of a novel comparative dataset of vehicle CAN-Bus extraction standards. The relevance of this section is to provide the authors and readers, an in-depth knowledge of the vehicle network and data associated with it, with the majority of this chapter published in [124]. As our overall focus turns to V2I, we use vehicle telemetry data as a use case. We also aim to create a specific message set for vehicle telematics in Smart Cities, so an understanding of vehicle sensor data, and ability to add logic to raw CAN data to create new parameters is imperative. To date, most literature relies on OBDII standard for capturing vehicle data. On the other hand, FMS standard is mostly used in Trucks and Buses, and only recently became available for small vehicles via a FMS gateway. Here, we capture data simultaneously using OBDII and FMS, creating a comparative dataset. The main objective of this research is to determine the best standard for capturing vehicle sensor data to use when monitoring fuel economy, CO_2 emissions, and other attributes that may affect quality of life in Smart Cities. The aim of this contribution is to address the following research question.

For vehicle telematics to be considered a key enabler in the development of Smart Cities, which in-vehicle data capturing technology provides the most accurate sensor data?

4.1.1 Hardware

The OBDII standard requires an OBDII adapter, which transmits data via Bluetooth to a smartphone app. The smartphone enriches the OBDII data with GPS information, and if required, other sensor data from the smartphone, such as accelerometer readings. The OBDII adapter used for this work was a high-end device called OBDLink MX. It addresses some of the known issues of other adapters, security in particular. Connection to this device requires a unique security scheme, which eliminates the risk of unauthorized access. The user requires physical access to enable Bluetooth pairing, making it hacker proof. The device is also advertised as the world's fastest Bluetooth OBDII adapter, up to 300% faster than other adapters, and can request up to 100 parameters a second, although this is far above the recommended 1 request per second. The adapter also comes with an android app that gives the user many options such as parameters to log and frequency of logging.

The FMS standard requires a CAN-Bus clip, and an FMS gateway, which is then connected to an Automatic Vehicle Location (AVL) device. The reason the gateway needs to be connected to an AVL is that there are no gateways on the market with GSM capabilities, so the data is transferred to an AVL, which enriches the incoming FMS data with GPS information before transmission. The AVL used for this project was a Falcom Fox-3, a highly capable device that can be configured to read OBDII, FMS and CAN information. The Fox-3 can be used as a standalone OBDII device, but for this purpose, it was configured to ingest FMS data from the gateway, geo-location enrichment and its GSM capability. Fox-3 configurations utilize a proprietary programming language called PFAL, which allows the user to request parameters from the FMS gateway, set the granularity of requests, and transmit to the Cloud.

The FMS gateway and CAN-Bus clip used are a product of Inventure Automotive, a global supplier of CAN data retrieving solutions. The gateway decodes CAN-Bus protocol parameters and translates them to FMS format. The CAN-Bus of vehicles are proprietary, meaning some CAN-Bus parameters are different. Therefore, you need to have the right decoding mechanism to eavesdrop and understand the communication of ECUs on the bus.

4.1.2 Installation

The OBDII adapter is easily installed as the OBDII port is visible and easily accessible. Once the adapter is plugged in, the user pairs the device to the smartphone via Bluetooth and is ready to go. There are numerous third party smartphone applications that are compatible with most adapters. The FMS option is more complicated and time consuming, requiring access behind the vehicle interior. The CAN-Bus clip needs to be directly connected to the CAN-High and CAN-Low wires, so professional assistance may be required to gain access to these wires, usually via the rear of the OBDII port, Figure 37. Once the clip is successfully fitted across the two wires, it is then connected to the FMS gateway Figure 38. Finally, the FMS gateway is connected to an AVL which is configured to transmit the data via GSM to the fleet manager’s servers, or in this case, an AWS server created by the author to ingest and store incoming vehicle data.

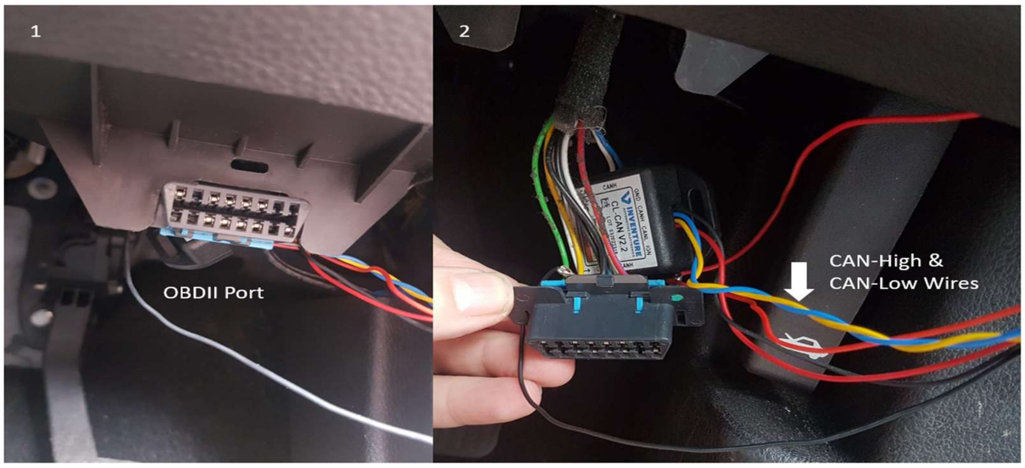


Figure 37: Accessing the Can Wires via the rear of the OBDII port. (CAN-High & CAN-Low wires are blue & yellow)

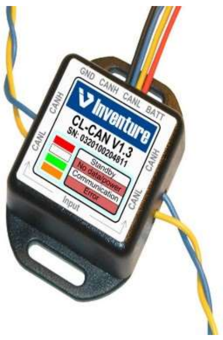


Figure 38: CAN-Clip sensing information on CAN-High and CAN-Low wires

Figure 39 provides an overview of the architecture used to extract data from the CAN-Bus via OBDII and FMS. The image shows the contactless clamp fixed directly to the CAN-High and CAN-Low wires, feeding data to the FMS gateway, which in turn feeds the data to the AVL. The OBDII adapter connects to the visible OBDII under the steering wheel and transmits via Bluetooth to the smartphone.

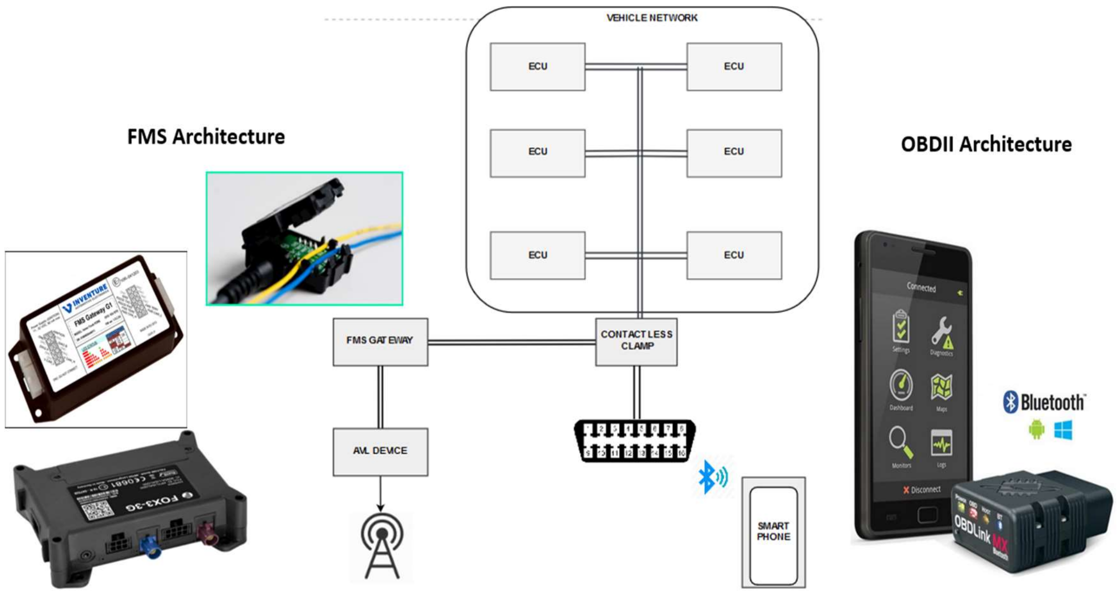


Figure 39: Architecture for receiving FMS and OBDII data from vehicle network

4.1.3 Data Generation

The vehicle used for testing was a 2011 Opel Astra. By requesting blocks of 15 parameters at a time, we could determine that, overall, 62 OBDII parameters returned data. As the recommended rate for requesting OBDII data is 1Hz (one request per second), the author was unwilling to request all 62 parameters at once, even though the OBDII adapter claims it can request 100 per second. For testing, 22 OBDII parameters were initially chosen due to their relationship with driving behaviour and fuel related parameters. As discussed, OBDII was initially developed for diagnostics, so many parameters can be considered unsuitable for this evaluation such as system vapour pressure, and battery related parameters etc. Other parameters that had no relationship to fleet management, or any previous literature in relation to fuel consumption were also ignored. Important parameters that were monitored were vehicle speed, RPM, throttle position, Mass-Air Flow (MAF), accelerator pedal sensors etc.

However, when requesting 22 OBDII parameters at a rate of 2Hz, there were issues with duplicate data and missed data. So the number was lowered to 7 parameters, as shown in Table 11. Reasons for choosing these parameters were based on their relationship with the FMS fuel rate parameter, previously used OBDII fuel algorithms, and their relevance in previous studies, as will be discussed. The relationships to FMS fuel rate was discovered utilizing Pearson's correlation.

Through this work, two datasets were made available to the research community. The reason for two datasets was due to granularity issues with OBDII. Dataset 1 consists of 22 OBDII parameters and 11 FMS parameters, recorded every second. Dataset 2 consists of 7 OBDII parameters and 11 FMS parameters, recorded every 500 ms.

Table 11: List of chosen OBDII and FMS Parameters at 2Hz

OBDII Parameters	**FMS Parameters**
Vehicle Speed (Km/H)	Vehicle Speed (Km/H)
Vehicle RPM	Vehicle RPM
MAF	Fuel Rate
Calculated Load Value	Accelerator position (%)
02 Sensor Lambda	Clutch Usage

Intake Air Temperature	Brake Usage
Drivers Demand- percent torque	Total Distance (Odometer)
	Total_Fuel_Used
	Vehicle Speed (cm/s)
	Fuel Level
	Engine Temp

A description of the OBDII parameters is as follows:

Vehicle Speed: Current speed of the vehicle in km/hr

Vehicle RPM: Current Revolutions per minute of engine.

MAF: Any OBDII-II compliant vehicle is equipped with MAF sensor (or a MAP sensor, which allows you to calculate MAF) [125]. MAF is a widely used parameter when determining fuel rate via OBDII, with much literature based on its involvement in determining instantaneous fuel economy.

Calculated Load Value: Indicates a percentage of peak available torque. Reaches 100% at wide open throttle at any altitude or RPM for both naturally aspirated and boosted engines.

02 Sensor Lambda: Determines the ratio between the amount of oxygen currently present in a combustion chamber and the amount that should be present to obtain perfect combustion.

Intake Air Temperature: Monitors the temperature of the air that is entering the engine of vehicle.

Drivers Demand - Percent torque: The requested torque output of the engine by the driver.

Intake Manifold Absolute Pressure: IMAP sensor is used to sense engine load. This information can be used to adjust ignition timing and fuel enrichment.

The FMS parameters are specifically designed for fleet management. Fuel Rate, accelerator pedal position, clutch and brake usage, Total Fuel Used, Total Distance (Odometer), vehicle speed and RPM, were tested and compared against the available OBDII parameters. For ground truth in our evaluation, through monitoring the vehicles odometer and fuel gauge before and after trips, we could accurately calculate the amount of fuel consumed per trip. This allowed us to evaluate the accuracy of the OBDII algorithms for determining fuel rate, and also the accuracy of the FMS Fuel Rate parameter. CO_2 emissions is also calculated. In relation to monitoring actual vehicle speed and stop/start events, the odometer was recorded during trips via a smart phone in a controlled environment. This gave us an accurate depiction of real-world information such as length of time the vehicle was stopped etc.

4.2 In-Vehicle Data Capturing Results

4.2.1 Basic Parameters

The main contribution of this section is to perform a comparison of both vehicle data extraction standards in terms of data accuracy, granularity, fuel efficiency and eco-driving monitoring. To the best of the author's knowledge, this will be the most in-depth comparison of standards to date. Comparing the basic parameters such as vehicle speed, RPM, engine temperature and fuel level was performed with little deviation between the standards. Table 12 shows the vehicle speed and RPM statistics recorded from OBDII and FMS over 2000 data points, equivalent to 1000 seconds.

Table 12: Comparison of OBDII and FMS vehicle speeds and RPM

Description	OBDII Speed (Km/H)	FMS Speed (Km/H)	OBDII RPM	FMS RPM
count	2000	2000	2000	2000
Mean	19.39	19.92	1348.86	1351.1
Std Dev	16.77	17.1	528.2	529.3
Min	0	0	707	716
25%	0	0	802	802
50%	21	22	1279.8	1276
75%	32	33	1815.5	1827
Max	57	58	3333	3290

At regular speeds, and over longer distances, both standards provide similar results, as shown above. The mean difference between standards for vehicle speed was 0.027 (2.7%) and RPM was 0.002 (0.2%). However, it was noticed on numerous occasions during testing, that when driving at very low speeds, OBDII has a tendency to drop to zero for longer periods, as shown in Figure 40 and Figure 41. This can hinder statistics, particularly in heavy traffic scenarios, as shown in Table 13 and Table 14. FMS also provides useful parameters such as clutch and brake usage, and accelerator pedal position (%). Clutch and brake parameters work on a binary basis. 1 means the pedal is in use, 0 means not in use. Bottom graph on Figure 40 and Figure 41 shows the added features FMS can offer over OBDII. Monitoring brake and clutch usage can give valuable insight into driving behaviour, particularly in stop and go traffic.

4.2.1.1 Scenario 1:

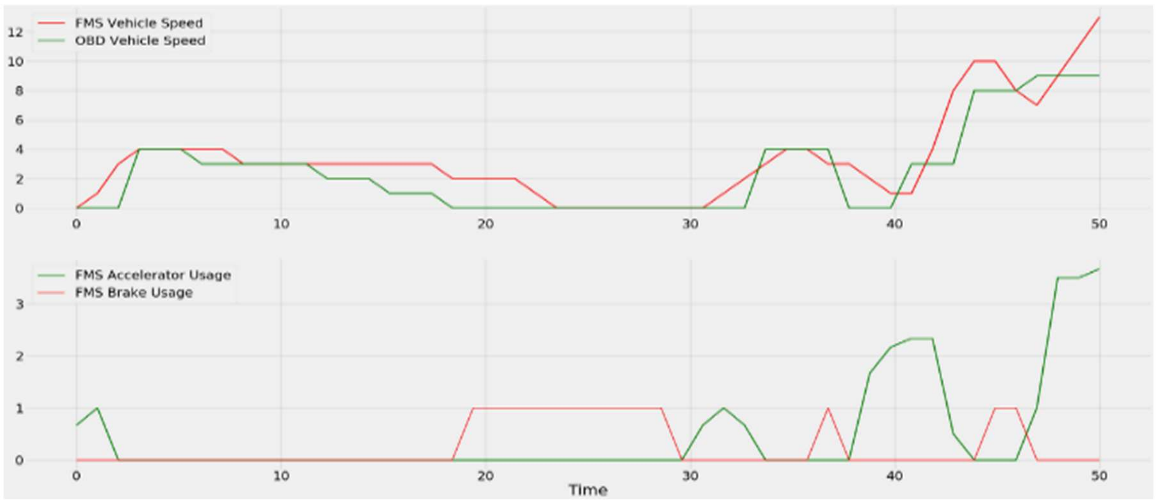


Figure 40: OBDII speed at zero for longer periods. Bottom graph shows extra FMS parameters. In the bottom graph, brake usage works on a binary basis, 1 when in use, and 0 when not in use. Accelerator usage is based on Accelerator pedal position (%)

Table 13: Statistical Analysis on FMS and OBDII speeds during scenario 1

Description	**OBDII Speed (Km/H)**	**FMS Speed (Km/H)**
Count	50	50
Mean	2.48	3.34
Std Dev	2.92	3.11
Min	0	0
25%	0	1
50%	2	3
75%	4	4
Max	9	13

4.2.1.2 Scenario 2:

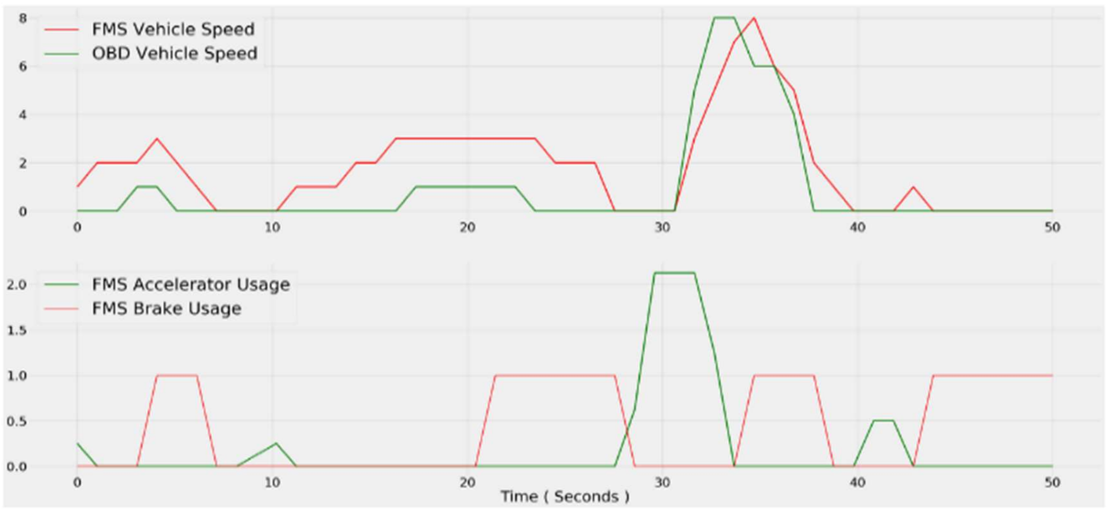


Figure 41: Another example of OBDII speed going to zero for a longer period. . In the bottom graph, brake usage works on a binary basis, 1 when in use, and 0 when not in use. Accelerator usage is based on Accelerator pedal position (%)

Table 14: Statistical analysis on OBDII and FMS Vehicle speed during scenario 2

Description	**OBDII Speed**	**FMS Speed**
Count	50	50
Mean	0.9	1.76
Std Dev	2.06	1.92
Min	0	0
25%	0	0
50%	0	1.5
75%	1	3
Max	8	8

In scenario 1, the difference between standards at lower speeds was 25.7%, whereas scenario 2 was 48.9%. This shows that OBDII is highly inaccurate in slow moving scenarios, which may not be desirable in monitoring traffic and driving behaviour in urban areas.

4.2.2 Data Granularity

As previously mentioned, FMS has a much higher rate of data available when compared to OBDII. Many FMS parameters are available at 10-100Hz. Due to limitations of the AVL device used for this work, we could only transmit parameters at a rate of 500ms, meaning FMS parameters, 11 in total, are monitored at 2Hz. The fastest we could request the 22 OBDII parameters was 1Hz, however, when reducing the number of parameters, we could request seven parameters at 2 Hz. It must be noted that this may be down to the test vehicle, as different vehicles may allow for more requests per second.

Figure 42 shows vehicle speed over a three minute period, monitoring OBDII data at 1Hz and FMS at 2Hz. Table 15 shows the analysis of Figure 42. Figure 43 and Table 16 represent RPM over the same trip.

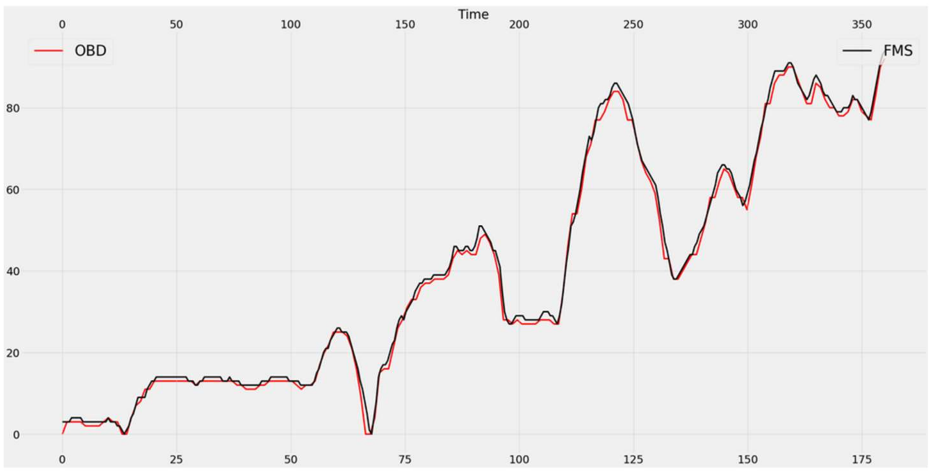


Figure 42: Vehicle Speed; FMS at 2Hz, OBDII at 1Hz over a 3 minute period

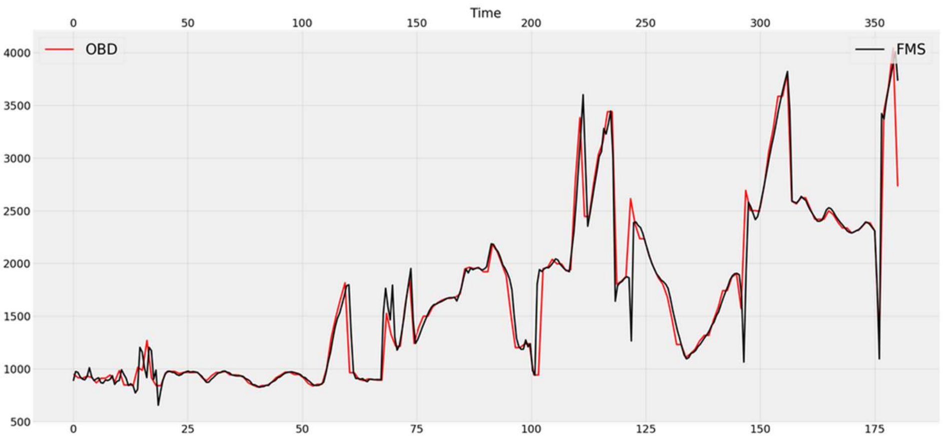


Figure 43: Engine RPM; FMS at 2Hz, OBDII at 1Hz over a 3 minute period

Table 15: Vehicle speed while FMS at 2Hz and OBDII at 1Hz over the same period

Description	OBDII Vehicle Speed	FMS Vehicle Speed
Count	180	360
Mean	39.228	39.95
Std Dev	28.548	28.591
Min	0	0
25%	13	14
50%	34	35
75%	64	65
Max	92	94

Table 16: RPM while FMS at 2Hz and OBDII at 1Hz over same period

Description	OBDII RPM	FMS RPM
Count	180	360
Mean	1661.692	1670.756
Std Dev	776.417	770.472
Min	826.5	654
25%	949.25	955.5
50%	1508.5	1572.5
75%	2234	2149.25
Max	4043.5	4010

The results above show slight variations, with a difference of 1.8% for vehicle speed and 0.5% for RPM. It may be fair to assume that monitoring basic vehicle parameters at

1Hz is as adequate as 2Hz. Similar testing showed very slight variations in monitoring fuel economy at 1Hz and 2Hz.

4.2.3 Fuel Economy

Fuel rate is a parameter made available on the CAN-Bus, (although rarely available via OBDII), given by the ECU. The fuel amount to the injectors is calculated by the ECU, by reading, for example, throttle position, engine speed, and vehicle speed. How much fuel is needed is calculated through the engine software. The ECU calculates the amount of fuel needed, and controls the injector opening times to inject the right amount of fuel [53].

Fuel economy is usually represented as the ratio of fuel consumed per distance travelled, being measured in terms of litres per 100 km (or in the U.S as MPG - miles per gallon). Calculating accurate fuel economy has been the focus of much research. This is due to its importance when considering cost savings and also environmental issues. To date, the majority of research has acquired OBDII parameters to calculate fuel economy. Further calculations can determine CO_2 emissions.

One OBDII parameter in particular, MAF, is used alongside constant parameters Air–Fuel Ratio (AFR) and Fuel density (FD), in numerous algorithms for determining fuel economy, Table 17. The ideal AFR for a complete combustion, is called stoichiometric AFR, and is 14.7:1 for gasoline, and 14.5:1 for diesel; meaning for every one gram of fuel, 14.5 grams of air are required for a perfect combustion. Fuel density is the weight of fuel in grams per litre.

Table 17: Fuel Information

Fuel Type	**Fuel Density**	**AFR Ratio**
Petrol	750g/l	14.7:1
Diesel	832g/l	14.5:1

Although there are OBDII algorithms that have been used regularly for calculating fuel economy [62], [126], [127], many of them are variations of the following two methods. In testing, both methods produce the same result.

Method 1:

Calculate Kilometres per litre with the following algorithm

$$\text{KmL} = \frac{AFR\ .\ FD\ .\ VS}{MAF\ .\ 3600}$$

Where KmL is kilometres' per litre. AFR is air fuel ratio, FD is fuel density, VS is Vehicle speed and MAF is Mass air flow rate. MAF is measured in g/s, so 3600 is used to convert to hours. This is then converted to fuel economy with the following.

$$\text{Fuel Economy (l/100km)} \quad = \frac{100}{KmL}$$

Method 2:

Calculate Litres per hour;

$$\text{LH} = \frac{MAF\ .\ 3600}{AFR\ .\ FD}$$

Fuel economy can then be determined using the following algorithm.

$$\text{Fuel Economy (l/100km)} \quad = \frac{LH\ .\ 100}{VS}$$

The authors in [128] proposed an enhanced version of Method 2 algorithm with two added OBDII parameters. The algorithm is shown as:

$$\text{LH} = \frac{MAF\ .\ \text{Loadcalc}\ .\ 3600}{AFR\ .\ (1+LTFT)\ .\ FD}$$

Where $\text{Load}_{\text{calc}}$ is calculated load (%), (OBDII PID 0104), and LTFT is Long Term Fuel Trim (PID 0107 or 0109). However, LTFT was unavailable on this test vehicle. The authors claim LTFT is optional, but may enhance the accuracy. During testing, this algorithm will be referred to as M2Enhanced.

As tests will show, Method 1 and 2 do not perform very well. This is due to the constant parameter AFR. In reality, engines will often vary from the ideal AFR. Therefore, there is an ideal AFR (14.5:1), and an actual AFR. The ratio between the actual air-fuel ratio (AFR_{actual}) and the ideal/stoichiometric air-fuel ratio (AFR_{ideal}) is called lambda (λ), which is reported as a standard OBDII PID 0124, or may be calculated from the commanded equivalence ratio parameter. When the air fuel ratio is exactly 14.5:1, $\lambda = 1$. However, throughout a trip, lambda will change. For example, an air-fuel ratio of 16:1 would translate to a lambda value of 1.088, using the algorithm:

$$\lambda = \frac{\text{AFR(actual)}}{\text{AFR(ideal)}}$$

4.2.3.1 Proposal 1: Lambda Algorithm:

Analysis on our dataset showed a high correlation between the 02 sensor, also called the lambda sensor, and FMS fuel rate. This was achieved through a Pearson's correlation algorithm against all parameters in the dataset. As FMS Fuel Rate is already considered to be an efficient parameter for monitoring fuel economy, any OBDII parameter with a correlation to this must be taken into consideration. The Lambda algorithm includes the "02 Sensor". Most car engines can determine how much fuel to expend into the engine based on the voltage of the O2 sensor. These sensors read the amount of unburned oxygen in the exhaust. The computer then uses this reading to balance the fuel mixture. As oxygen content in the exhaust increases (known as a lean condition) the sensors voltage reading decreases. This signals the computer to increase the amount of fuel the injectors are delivering. A decrease in oxygen content is known as a rich condition. The oxygen sensor voltage increases as a result of this richening, and the computer reacts by reducing fuel flow. This process is continuous as long as the engine is running. The proposed Lambda algorithm is as follows:

$$\text{Fuel Economy (l/100km)} \;=\; 100\,\frac{MAF\,.\,C}{\lambda\,.\,\text{AFR}\,.\,\text{VS}}$$

Where C is a constant (3.6) that converts MAF from g/s to l/h, AFR is Air–Fuel Ratio and λ is the 02 Sensor value.

4.2.3.2 Proposal 2: The D_Demand Algorithm:

Whilst running statistical analysis on our dataset, a high correlation between "Driver's demand engine - percent torque", (OBDII PID 0161) and the proposed Lambda algorithm was discovered. It was noticed that the output from the Lambda algorithm was a correlating percentage of the Drivers Demand output (e.g 20%). Furthermore, this percentage had a strong correlation with the IaT parameter, for example, when the Lambda algorithm output was 20% of the Drivers Demand output, the IaT parameter was 20° Celsius. Previous work such as [129] has shown that Intake Air Temperature (IaT) can be an important factor in fuel economy. Further testing allowed us to create the following algorithm by including both parameters. Calculating IaT as a percentage of Drivers Demand, i.e.; (DD /100 * IaT), resulted in an adequately performing fuel economy prediction algorithm. As the Drivers demand OBDII parameter often returns negative numbers, all parameters in this proposed algorithm were squared to negate this, before using the square root as the final output. Further analysing the output of this algorithm when comparing it to Proposal 1, we discovered adding 1 to the output brought it much closer to Proposal 1 and FMS fuel rate algorithm. The final algorithm is as follows:

$$\text{Fuel Economy (l/100km)} = 1 + \sqrt{D^2 \, . \, IaT^2}$$

Where D is "Driver's demand engine - percent torque" divided by 100, and IaT represents Intake Air Temperature.

4.2.3.3 Calculating Actual Fuel Economy

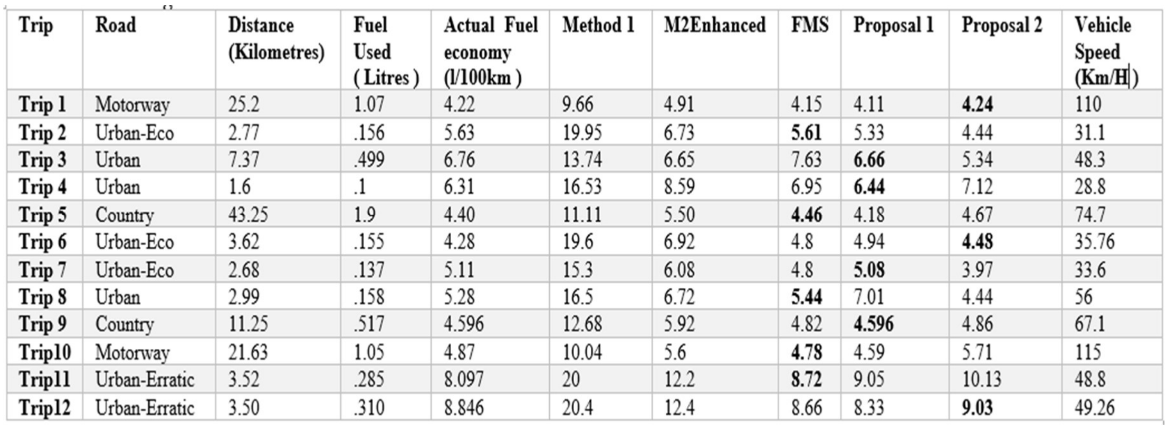
FMS has two parameters which allowed us to accurately test the algorithms. Total_Vehicle_Distance (Odometer) and Total_Fuel_Used are readily available via FMS.

This allowed us to calculate how much fuel was used over any given distance, which can then be converted into an actual l/100km value. This was calculated with the following:

$$\text{Actual (L/100km)} = 100\,\frac{\text{TF} - \text{TS}}{\text{OF} - \text{OS}}$$

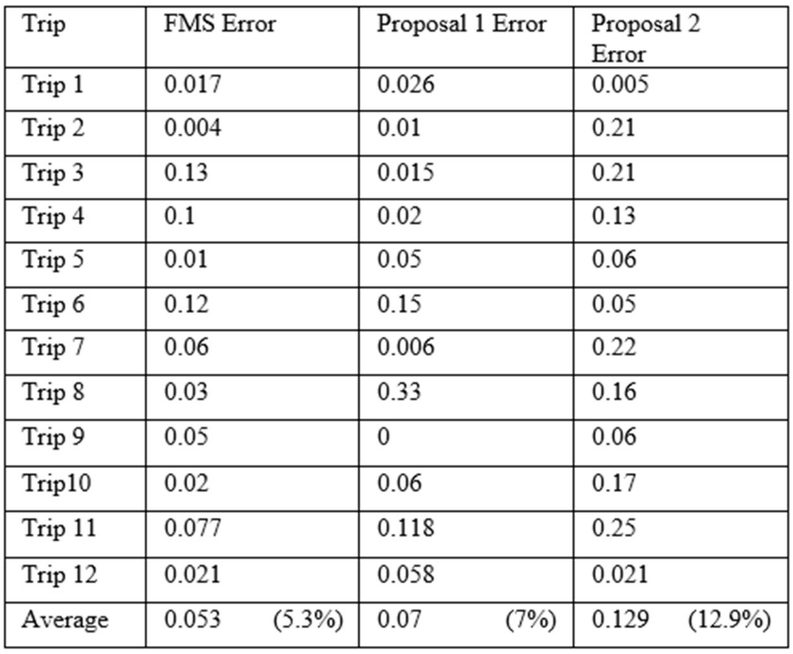
Where $T_F - T_S$ is Total_Fuel_Used at the end of trip, minus Total_Fuel_Used at start of trip. O_F - O_S is Odometer reading at end of trip minus reading at start of trip.

Table 18: Calculated mean algorithm results over multiple trips.



Trip	Road	Distance (Kilometres)	Fuel Used (Litres)	Actual Fuel economy (l/100km)	Method 1	M2Enhanced	FMS	Proposal 1	Proposal 2	Vehicle Speed (Km/H)
Trip 1	Motorway	25.2	1.07	4.22	9.66	4.91	4.15	4.11	**4.24**	110
Trip 2	Urban-Eco	2.77	.156	5.63	19.95	6.73	**5.61**	5.33	4.44	31.1
Trip 3	Urban	7.37	.499	6.76	13.74	6.65	7.63	**6.66**	5.34	48.3
Trip 4	Urban	1.6	.1	6.31	16.53	8.59	6.95	**6.44**	7.12	28.8
Trip 5	Country	43.25	1.9	4.40	11.11	5.50	**4.46**	4.18	4.67	74.7
Trip 6	Urban-Eco	3.62	.155	4.28	19.6	6.92	4.8	4.94	**4.48**	35.76
Trip 7	Urban-Eco	2.68	.137	5.11	15.3	6.08	4.8	**5.08**	3.97	33.6
Trip 8	Urban	2.99	.158	5.28	16.5	6.72	**5.44**	7.01	4.44	56
Trip 9	Country	11.25	.517	4.596	12.68	5.92	4.82	**4.596**	4.86	67.1
Trip10	Motorway	21.63	1.05	4.87	10.04	5.6	**4.78**	4.59	5.71	115
Trip11	Urban-Erratic	3.52	.285	8.097	20	12.2	**8.72**	9.05	10.13	48.8
Trip12	Urban-Erratic	3.50	.310	8.846	20.4	12.4	8.66	8.33	**9.03**	49.26

Table 19: Error% of FMS Fuel Rate & algorithms compared to the actual fuel economy



Trip	FMS Error	Proposal 1 Error	Proposal 2 Error
Trip 1	0.017	0.026	0.005
Trip 2	0.004	0.01	0.21
Trip 3	0.13	0.015	0.21
Trip 4	0.1	0.02	0.13
Trip 5	0.01	0.05	0.06
Trip 6	0.12	0.15	0.05
Trip 7	0.06	0.006	0.22
Trip 8	0.03	0.33	0.16
Trip 9	0.05	0	0.06
Trip10	0.02	0.06	0.17
Trip 11	0.077	0.118	0.25
Trip 12	0.021	0.058	0.021
Average	0.053 (5.3%)	0.07 (7%)	0.129 (12.9%)

Table 18 shows the tested algorithms over multiple trips and driving types. The trips include Motorway, Urban, and Country roads. Driving styles such as Urban-Eco, in which the driver utilized eco-driving rules, and Urban-Erratic, in which the driver aggressively used the accelerator and Brake pedals, were also monitored. The actual distance travelled and actual fuel used is included, which allowed us to calculate actual fuel economy. The results of the algorithms represent the average fuel economy over each trip.

Table 19 shows the error rate between the actual fuel economy and the FMS, proposal 1 and proposal 2 (the three best performing) algorithms over 12 trips. FMS has an average error of just 5.3%, closely followed by proposal 1 at 7%, and proposal 2 at 12.9%.

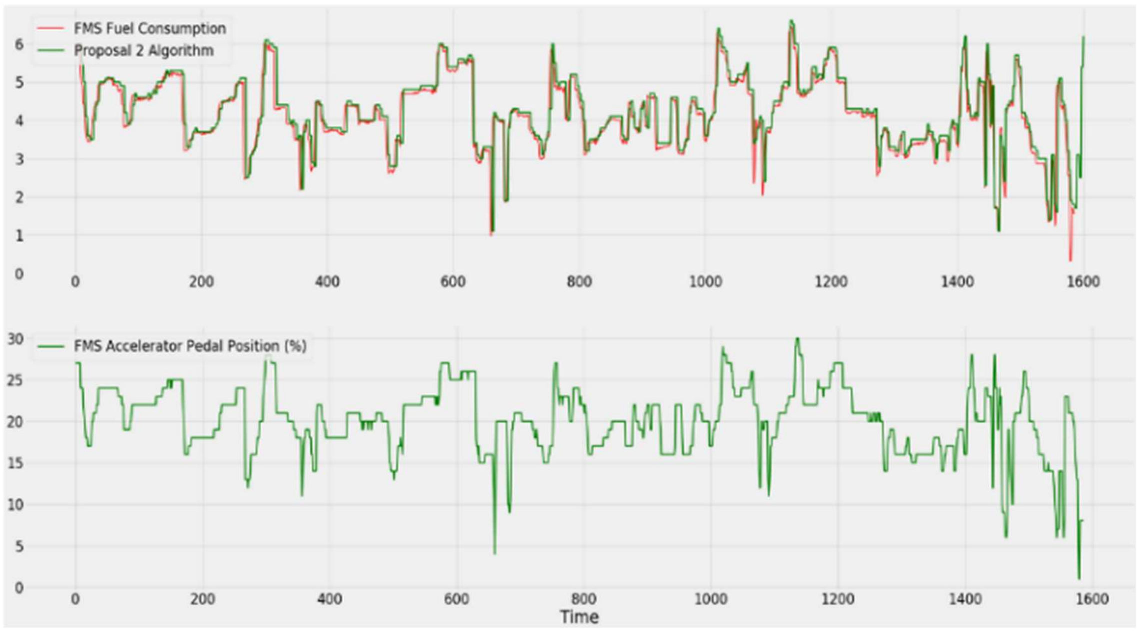


Figure 44: Top graph showing similarity of Proposal 2 and FMS fuel parameter

Figure 44 shows the correlation between the best performing algorithms in Trip 1, and the accelerator pedal position. This shows the clear influence the accelerator pedal has on fuel economy. In this example, there was a 0.984 correlation between FMS fuel rate and the accelerator pedal position. There was a 0.856 correlation between Proposal 2 and the accelerator pedal position.

Similarly in Trip 2, an urban trip, there was a strong correlation between the accelerator pedal position and the best performing fuel algorithms, as shown in Figure 45. There was a correlation of 0.92 between FMS fuel economy and the accelerator pedal position. While there was a correlation of 0.91 between Proposal 1 and the accelerator pedal position.

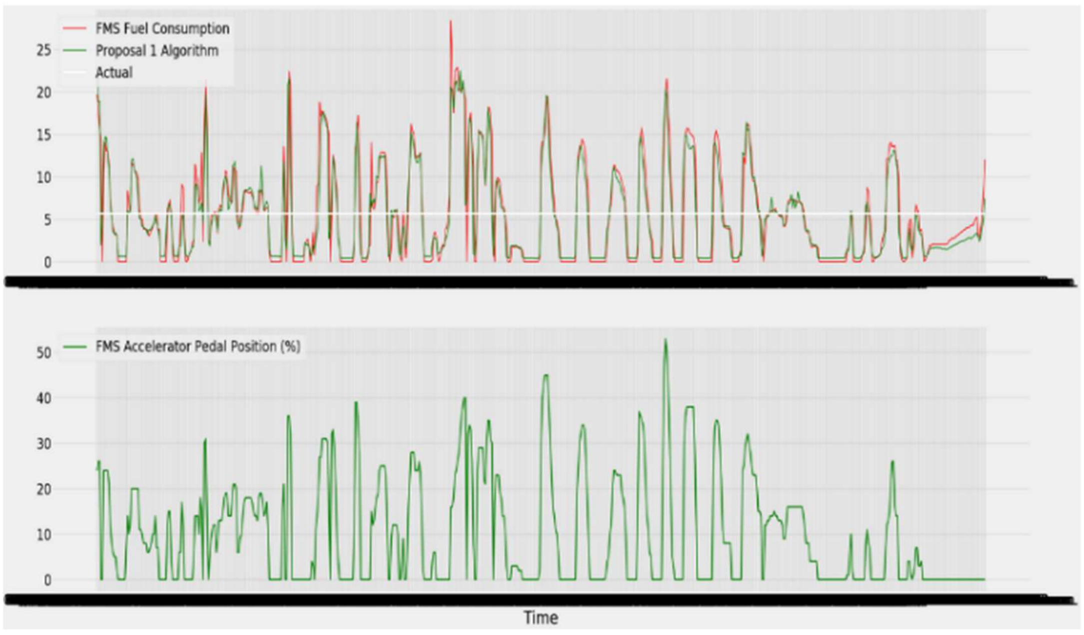


Figure 45: Top graph showing similarity between Proposal 1 and FMS fuel.

4.2.4 Acceleration/Deceleration

Acceleration and deceleration play a major factor in monitoring driver behaviour. Hard acceleration and deceleration have been defined as an increase/decrease of 1.4705 m/s^2 when driving [130]. The authors of [131] collected data using five passenger cars of different sizes and performances; the cars were equipped with a data-logging system. The data were used to determine the main properties that affect emissions and fuel use. They found that the percentage of time when acceleration exceeded 1.5 m/s^2 was one of the most important parameters, and they considered such states as representative of extreme acceleration. During testing, a number of hard accelerations/decelerations were taken place in a controlled environment. Results were monitored by converting the OBDII vehicle speed from km/H to m/s, while FMS has a parameter that gives speed in cm/s. Hard Acceleration/deceleration could then be monitored using the following equation.

$$A = (ms - ms.shift > 1.4705).sum$$

Where A is an occurrence of aggressive driving behaviour, ms is the current value for m/s^2 and ms.shift is the value of m/s^2 one second previous. As expected OBDII and FMS provided similar results. In five scenarios, the results are as shown in Table 20.

Table 20: Monitoring Aggressive driving behaviour

Scenario	OBDII No. of hard acceleration/decelerations	FMS No. of hard acceleration/decelerations	Actual No. of hard accelerations/decelerations
Test 1	36	38	38
Test 2	30	30	30
Test 3	27	27	27
Test 4	30	31	31
Test 5	19	19	19

4.2.5 CO_2 Emissions Calculations

This section focuses on CO_2 emission calculations from the aforementioned fuel algorithms. CO_2 emissions are calculated from amount of fuel used. As previously mentioned, a litre of diesel weighs 832g. Diesel consists of 86.2% carbon, meaning there are 717g of carbon per litre of diesel, which requires 1911g of oxygen to combust. Therefore, there is 2628g of CO_2 per litre of diesel [44]. With this information, and the FMS fuel used parameter, the actual CO_2 emissions per trip can be calculated. However, OBDII does not have a Fuel Used parameter, so using the fuel algorithms, we can estimate the average CO_2 emissions in Kg per 100Km using the using the following:

$$CO2_{avg} = CO2_{PL} . FE$$

Where $CO2_{PL}$ is the amount of CO_2 per litre (2628g) and FE is the calculated Fuel Economy from the algorithms. From here, we can calculate CO2 emissions per trip, using:

$$CO2_{trip} = \frac{CO2_{avg}}{100} \; . \; AD$$

Where AD is actual trip distance.

Table 21 shows the calculated average CO_2 emissions per 100km during Trip 1. Also shown is the calculated CO_2 emissions over the trip. As a note, the test vehicle specification manual states CO_2 emissions for this particular make and model, as 109 g/km (10.9kg over 100km). Figure 46 shows the results of our calculations from trip 1. To improve the visualization of the other results, Method 1 is not inserted into Figure 46, Figure 47, and Figure 48 as it has already been established it is highly inaccurate.

Table 21: CO_2 emissions (per 100km) from fuel algorithms during Trip 1

Algorithm	**Fuel Economy (l/100km)**	**$CO2_{avg}$ (KG) per 100km**	**$CO2_{trip}$ (KG) per trip**
Method 1	9.66	25.39	6.39
M2Enhanced	4.91	12.9	3.25
FMS	4.15	10.91	2.75
Proposal 1	4.11	10.8	2.72
Proposal 2	4.24	11.14	2.81
Actual	4.22	11.09	2.79

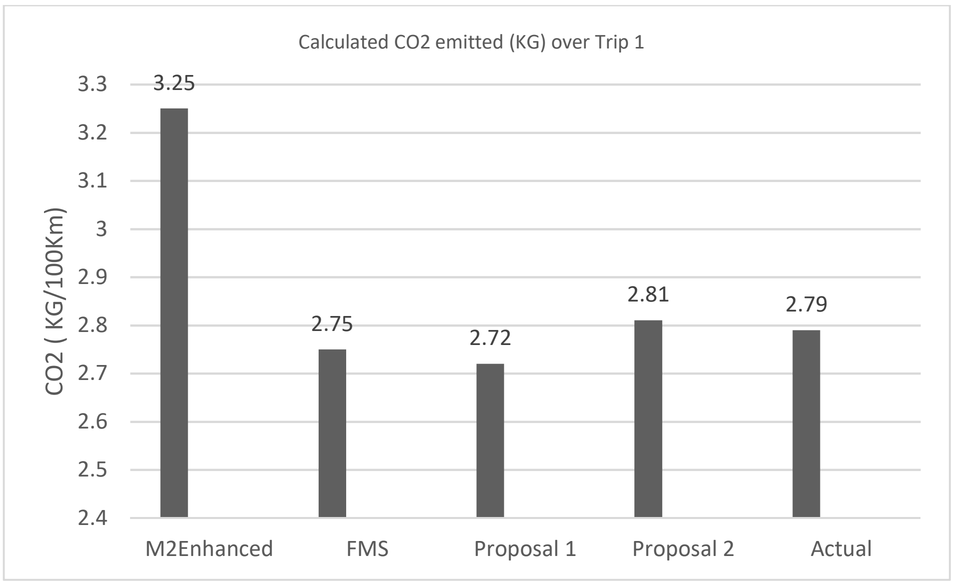


Figure 46: Calculated CO_2 emissions from trip 1 (Motorway).

Table 22 and Table 23 show the difference in CO_2 emissions when driving erratically. When comparing actual CO_2 emissions per 100km from both tables, an increase of 43.78% in CO_2 emissions can be seen when aggressive accelerating/decelerating is performed. Figure 47 and Figure 48 show the results in graph format.

Table 22: CO_2 Calculations from fuel algorithms from trip 2 (Urban-Eco)

Algorithm	**Fuel Economy (l/100km)**	**CO2 emissions (KG) per 100km**	**CO2 emissions over trip (KG)**
Method 1	19.95	54.43	1.51
M2Enhanced	6.73	17.69	0.49
FMS	5.61	14.74	0.408
Proposal 1	5.33	14.01	0.388
Proposal 2	4.44	11.69	0.324
Actual	5.63	14.8	0.410

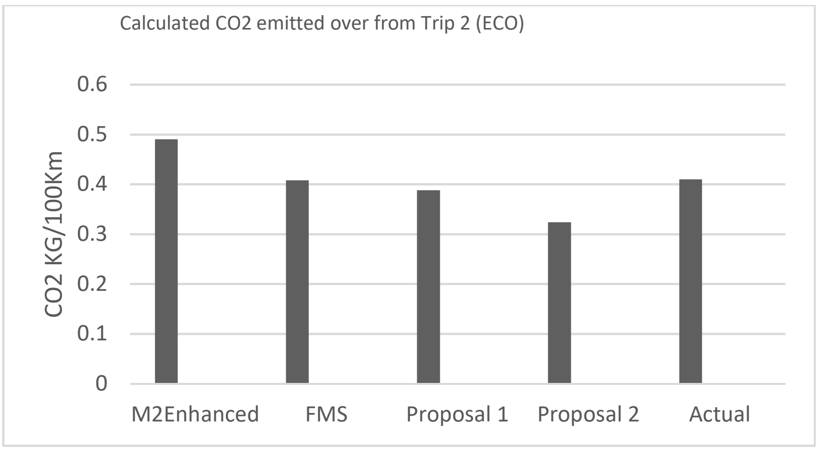


Figure 47: CO_2 emissions from trip 2 calculated from algorithms - Eco

Table 23: CO_2 Calculations from fuel algorithms from trip 11 (Urban-Erratic)

Algorithm	**Fuel Economy (l/100km)**	**CO2 emissions (KG) per 100km**	**CO2 emissions over trip (KG)**
Method 1	20	52.56	1.85
M2Enhanced	12.2	32.1	1.13
FMS	8.72	22.92	0.807
Proposal 1	9.05	23.78	0.837
Proposal 2	10.13	26.62	0.937
Actual	8.097	21.28	0.748

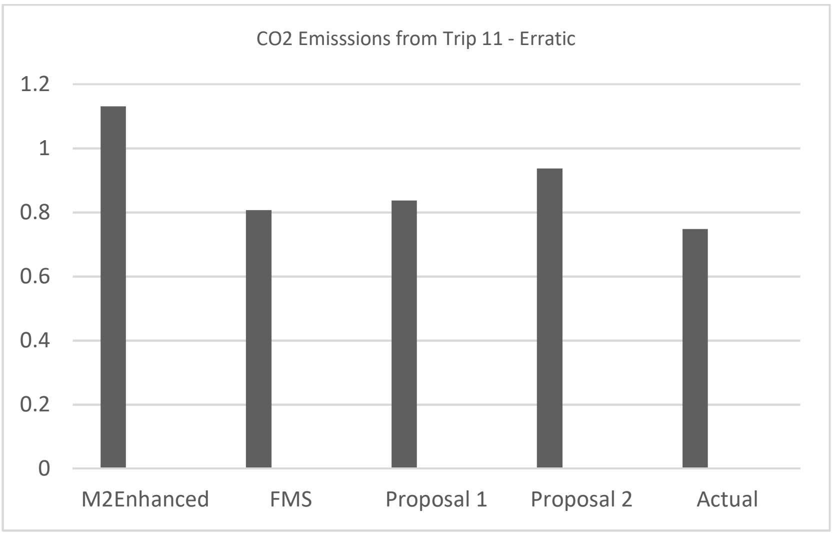


Figure 48: CO_2 emissions from trip 11, based on fuel algorithms (Erratic driving)

4.3 Conclusion

This section presented an evaluation of OBDII and FMS standards. While testing individual parameters such as vehicle speed and RPM, there is little variation between both standards. However, OBDII performs poorly at lower speeds, with a tendency to drop to zero in stop and go traffic. In urban areas, this can greatly affect data analysis, with an error rate of up to 48% in our tests. Fuel usage is the most important parameter in terms of Eco-Driving. An accurate algorithm is imperative for efficiently monitoring fuel economy, as incorrect data can greatly skewer calculations regarding driving behaviour and CO_2 emissions. In this section we introduced two novel OBDII algorithms that improve OBDIIs performance for fuel monitoring, however, we have shown FMS to be slightly more accurate than OBDII for monitoring Fuel usage and CO_2 emissions. Another point to consider is OBDII greatly relies on the availability of specific parameters, which will not always be the case.

FMS also holds an advantage over OBDII with the extra parameters such as clutch and brake usage, accelerator pedal position for monitoring driver behaviour, and Total Fuel Used and Total Distance Travelled for monitoring actual fuel economy. FMS also outperforms OBDII in terms of data granularity, without adding extra load on the vehicle network. Overall this study showed that OBDII is capable of monitoring driver behaviour, however, FMS would be considered the more accurate, reliable source of CAN-Bus information.

5 WAVE-Flow

5.1 Architectural Design & Methodology

To date, our focus has solely been on Edge-Computing, and an Edge to Cloud paradigm. As our focus turns to Vehicle-to-Infrastructure, additional requirements and mechanisms need to be included into our DAGLADs architecture. RSUs play a central role in V2I, acting as the intermediary device between vehicles and surrounding infrastructure and internet connectivity. As discussed throughout the literature, the majority of works define RSUs as gateway devices. This section introduces our proposed architecture WAVE-Flow. The main goal of WAVE-Flow is to enhance communication between OBUs and RSUs, while improving the level of governance RSUs can play in V2I. This chapter has been published in [132] and work submitted [133]. There is a wealth of literature that focus on improving data dissemination in vehicle scenarios. However, the vast majority focus on communication and routing protocols. Our goal is to take a more data-centric approach, focusing on inner mechanisms of the WAVE devices.

One of the novelties of WAVE is that higher layer applications can determine lower layer parameters such as data rate, transmission power etc. However, this study brings together the concepts of Flow Based Programming and WAVE with the aim of introducing a novel concept in which higher layer applications of the RSU can determine lower layer parameters of the vehicles. Through FBP, WAVE, and our previous focus on bi-directional communication, our aim is to give RSUs governance over the Connected Vehicles in the area as shown in Figure 49.

In the literature review we mentioned EDCA and the RTS/CTS mechanism, and also the modifying of data rates to enhance packet delivery. Taking this into consideration, and to be best of the author’s knowledge, we will be the first to evaluate how a combination of QoS mechanisms, modifying data rates and packet inter-arrival times may reduce packet loss and collisions while increasing throughput in our simulations.

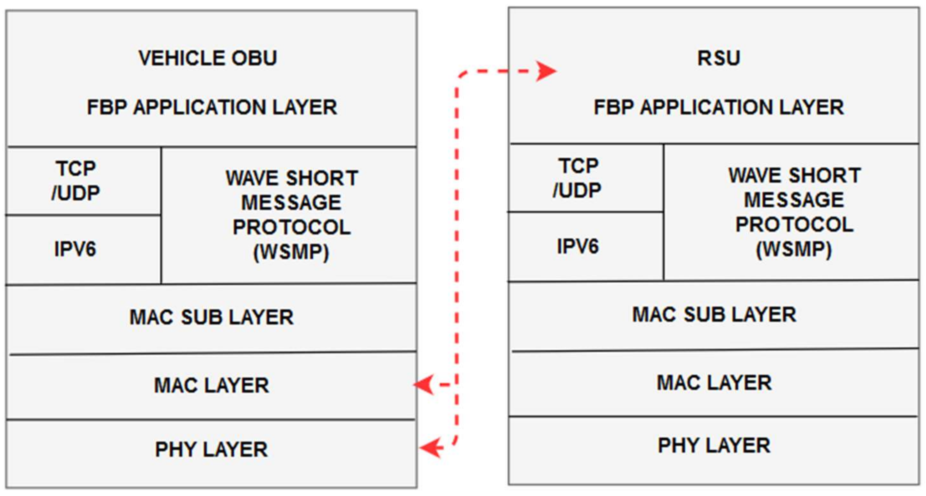


Figure 49: Higher Layer of RSU governing Lower Layer Parameters of OBU

Connected Vehicles will most likely incorporate multiple communication protocols in future. Because of existing Wi-Fi hotspots and the upcoming rollout of 5G throughout a Smart City, it is our belief that WAVE will not be the only communication medium utilized in V2I. With this in mind, we introduce and evaluate an in-vehicle mechanism, W-V6, that dynamically changes the medium in which data is transmitted, dependant on the vehicles location.

As a use case for WAVE-Flow, we discuss two scenarios, a multi-user fleet management system, and RSU governance. Building on our current architecture DAGLADs and previous literature, the aim of this section is to address the following research question.

Can a Multi-Tier Flow Based Programming Architecture enhance the distribution and governance of vehicle sensor information in Smart Cities?

5.1.1.1 Estinet

Deployment and field testing of V2V and V2I is highly costly and requires intensive labour. As much of the hardware is still not mainstream, a practical alternative is the use of simulation software to evaluate the performance in a wide variety of scenarios. Simulating VANET is notoriously difficult due to the priority coding and the need for coupling multiple applications such as network simulators and mobility generators. There are numerous network platforms such as NS2, NS3 and OMNET that can be integrated with other applications to allow for the simulation of VANET. There are also a small number of platforms that integrate the mobility and network applications into one. This allows for quicker observations of parameter and scenario modification.

To test our scenarios, we use EstiNet Network Simulator, formerly known as NCTUns [134], [135], [136]. EstiNet is a novel commercial-grade network simulation/emulation platform with V2V, V2I and WAVE capabilities. To simulate vehicular traffic, EstiNet first supports road-building functions so that cars can move on roads. A road network can be built by hand or by importing a roadmap file. Secondly, human driving behaviour is applied in a car to control its movement, such as car following, lane changing, overtaking, and compliance with traffic light signals. EstiNet 10 is the most recent version and was used for our simulations.

There is also a choice of nodes available, such as Road-Side Units, 802.11p vehicles, ad-hoc vehicles using 802.11a/g/n, and also multi-interface vehicles. Custom user configuration files can be implemented on the nodes, determining what data type, data size and rate of transmission. As an alternative, real data can be transmitted between nodes by pointing the file location to the nodes. Additional modules such as FIFO (queueing), flow classification (QoS) can be configured and added to the nodes protocol stack.

Communication between nodes can be made visible as shown in Figure 50. The user also has the option to configure nodes with a visual transmission range as shown in Figure 51. This allows you to view which nodes are in communication range of one another (monitoring the hidden node issue). With the above-mentioned features, EstiNet can simulate realistic traffic situation on roads. IEEE 802.11p, IEEE 1609.3, and IEEE 1609.4 protocols are supported by EstiNet to simulate V2V/V2I networking and communication. These protocols are designed to support WAVE with lower transmission latency and higher transmission quality under high-speed movement situation.

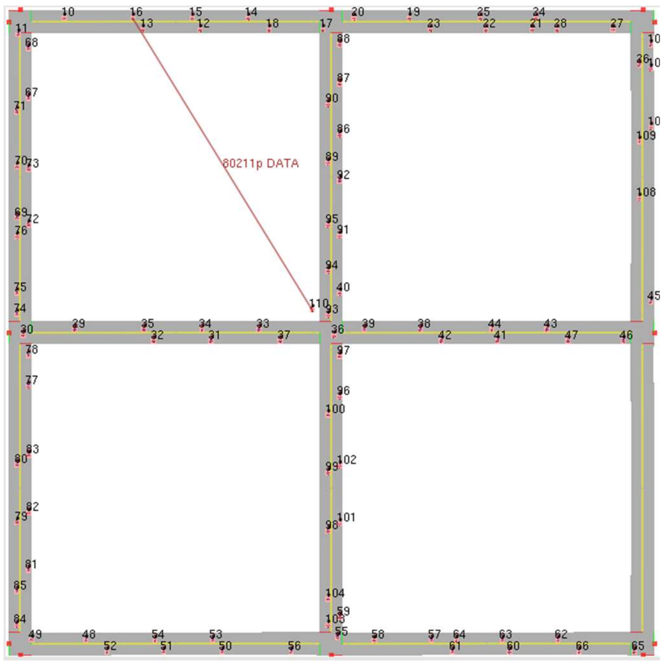


Figure 50: 3 x 3 grid simulation with 100 vehicles and one RSU (Node 110).

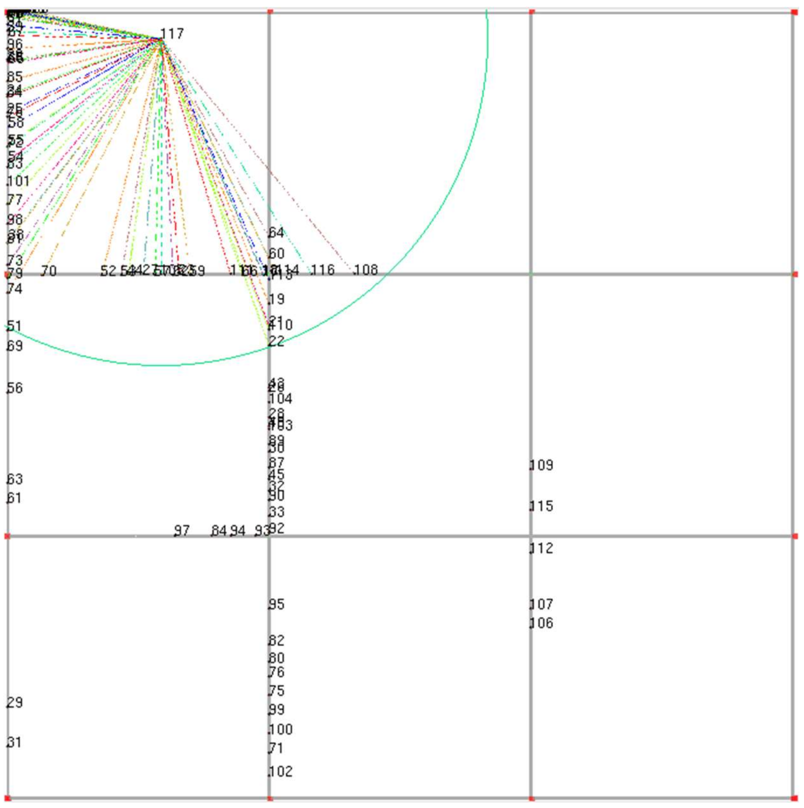


Figure 51: 4x4 grid... RSU (Node117) transmission range visible via green circle

5.2 WAVE-Flow Components

WAVE-Flow spans across three tiers; the Edge node (Vehicle), Fog node (RSU) and Central node (Server), Figure 54. The central node may be in the Cloud or the city developer's office. It is a device or group of devices with large processing and storage capabilities. Here, developers monitor large quantities of incoming data, allowing them to create and distribute new protocols and algorithms instantaneously.

The middle tier, implemented on the RSUs, also has processing and storage capabilities, although not as much as the central node. The RSU has bidirectional communication with the central node and other RSUs via Ethernet, and the Connected Vehicles via WAVE. The final tier of the architecture is situated on the vehicles. The FBP technology allows for immediate local processing on vehicle sensor data. The addition of a listening server allows for incoming protocols and algorithms from higher tiers to be implemented immediately. WAVE-Flow primarily focuses on vertical communication between nodes as there is an existing large body of work that presents horizontal techniques such as data aggregation between clusters of vehicles. In essence, V2V can be seen as horizontal communication. Figure 52 provides a visual overview of WAVE-Flow and its components.

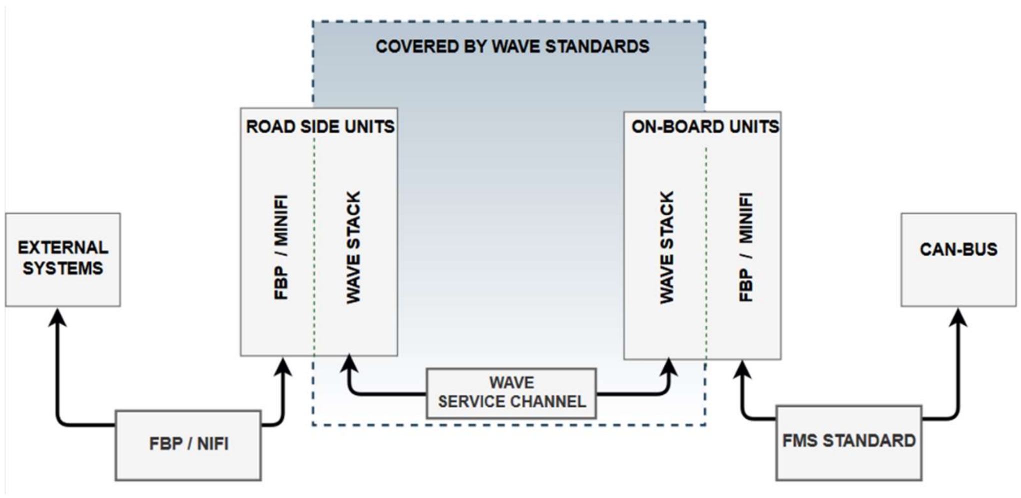


Figure 52: Overview of WAVE-Flow

5.2.1 *Provider Service Context (PSC)*

Although V2V, V2I and WAVE mostly operate under a broadcasting system, adding extra information to the WSA allows us to overcome this issue. As previously stated, we aim to utilise the Provider Service Context (PSC) parameter. PSC is optional and can be used to provide additional information about a service. To date, there is no literature in academia or industry using the PSC parameter in a V2V/V2I scenario. As discussed, vehicles are configured to respond to specific PSIDs in a WSA. However, the PSC allows you to add more context to the WSA. We aim to utilize PSC to further filter out vehicles that will switch to the service channel.

5.3 Use-Case I: Multi-User Fleet Management System

In this scenario, we focus on vehicular information for non-safety application usage. The vehicles collect and aggregate copies of frequently transmitted safety beacons, enriching them with parameters suited to vehicle telematics, and transmits when requested from the RSU. RSUs are connected to one another and also the central server via Ethernet. Each RSU transmits the incoming messages directly to the central database. However, as is the versatility of our architecture, the RSU also has the capacity to aggregate the messages received from all vehicles in its vicinity, giving a snapshot of current driver and traffic behaviour in the area. This information can be distributed to other RSUs, leading to overall improvement in mobility in the city.

5.3.1 Advanced Message Set (AMS)

In the previous chapter we looked at vehicle telemetry data and the valuable information it possesses in aiding fleet managers, and also city developers. With this in mind, we extend the WAVE standard with an Advanced Message Set (AMS) which consists of raw CAN data and newly created parameters, enriched with GPS information. AMS is designed to provide valuable information with regards to monitoring current driver and traffic behaviour. The AMS is transmitted from the vehicles to a nearby RSU, which transmits the messages to a centralized database. This gathered information may provide insight to city developers when monitoring and improving urban mobility.

Safety messages are beacons of information each vehicle transmits every 100ms in a V2V scenario. These small frames contain information such as direction of travel, GPS, speed, vehicle weight and length. They are highly advantageous for improving safety in urban areas. However, these small messages are simply discarded once transmitted. Our goal is to store a copy of the messages locally, aggregate and enhance them with other parameters that may benefit non safety applications, such as fleet management.

Although beacons are transmitted every 100ms for safety reasons, our proposed AMS will be transmitted less frequently, between 1-20 seconds. This is in line with most current fleet management systems that only require their data in soft real-time. Two parameters that distinguish fleets and individual vehicles within the fleet are also added. A unique fleet ID is assigned to the vehicles during initial configuration. Each vehicle of the same fleet will have the same fleet ID, but different vehicle ID, the Vehicle Identification Number (VIN) that is accessible from the vehicle network. Other parameters relevant to fleet management are included, such as average fuel consumption in Litres per 100km (L/100km), average RPM, number of hard acceleration/braking, and engine idle time. These parameters add extra insight into driver behaviour and efficiency. As shown in Figure 53, some parameters are created by applying logic to multiple raw CAN parameters. Table 24 shows the message set of the AMS.

Table 24: Proposed message set of the AMS

Unique Fleet ID	Current Acceleration
Vehicle ID (VIN)	Calculated CO_2 Emissions
Start Timestamp	Vehicle Size
Stop Timestamp	Avg. Fuel Consumption
3D Position Start	No. of Hard Accelerations
3D Position Finish	No. of Hard Braking
Average Speed	Engine Idle Time
Current Heading	Avg. Revs Per Minute
Current Steering Wheel Angle	No. of Excessive Wheel Turns

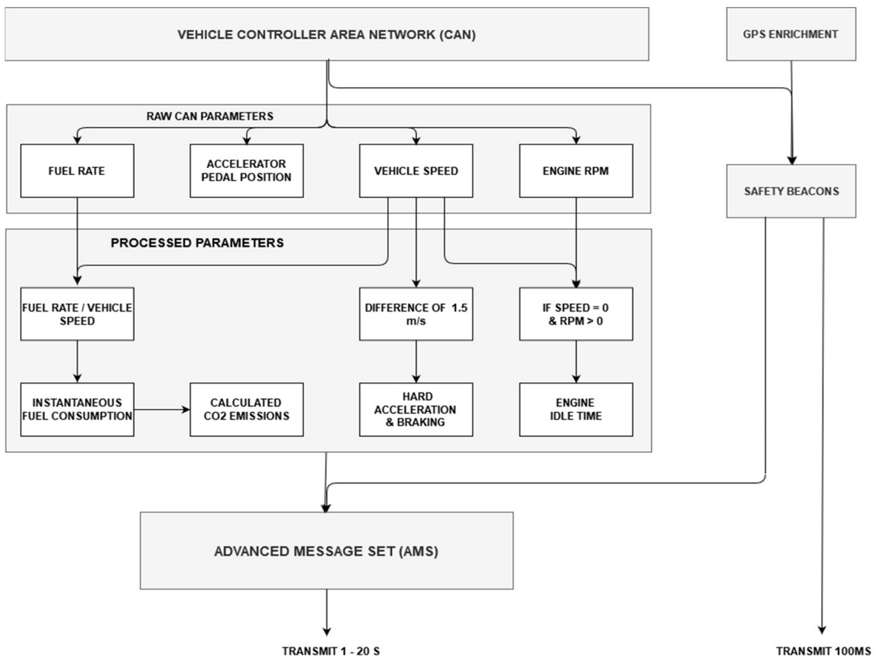


Figure 53: Creation of AMS from CAN and GPS

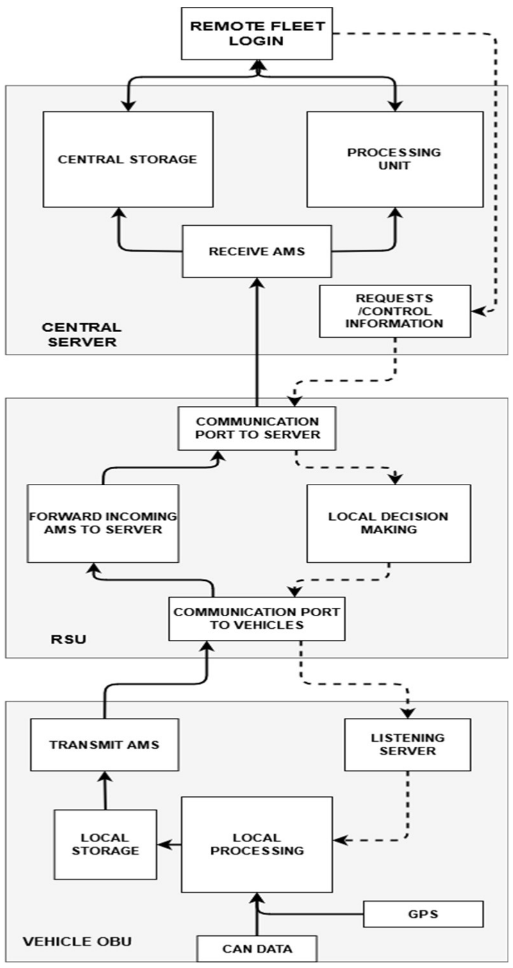


Figure 54: Three Tier Implementation Architecture

5.3.2 *Unique Fleet ID*

Public and private fleets can avail of our proposed platform via a unique ID tagging system. For example, emergency services may be assigned a unique fleet ID; the vehicles within this fleet attach the unique fleet ID to their message before transmission to the RSU. The fleet manager can then access only their associated vehicle data from the central database via their unique fleet identification number. WAVE-Flow also allows fleet managers to communicate directly with their vehicles via RSUs; distributing new fleet IDs, modifying the subset of transmitted data, requesting diagnostic codes etc. In essence, RSUs can be seen as control information “hotspots” throughout a city, as represented in Figure 57.

Figure 55 shows a snippet of a WSA with the PSC parameter configured with a unique fleet ID. Vehicles of this fleet will switch to service channel 172 and exchange information with this WSA, other vehicles will ignore.

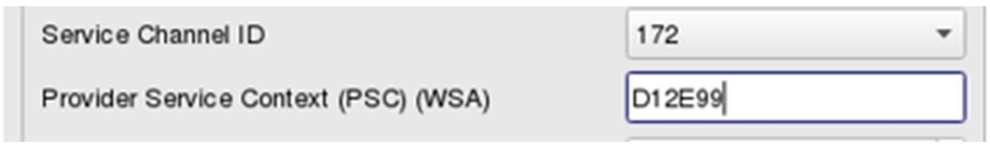

Service Channel ID: 172
Provider Service Context (PSC) (WSA): D12E99

Figure 55: Example of PSC configured with Unique Fleet ID

Up to 15% of city traffic is freight (trucks and vans belonging to a fleet), add in public transport and emergency services, this number is close to 20%, which would be a large enough percentage of overall traffic to give city developers a good indication of current mobility and traffic congestion [137] . We utilize this percentage to test our architecture, for example, 300 vehicles communicating with an RSU represents 1500 vehicles in the area. This is often referred to as using probe vehicles, or floating car data (FCD) to collect traffic information [138]. As previously mentioned, individual fleets may access their data via a unique fleet ID, however, city authorities may request all fleets to respond to a universal ID, “FFFF” (this will be the most common request). This acts similarly to a broadcast message, requesting all fleets in the area to transmit their information to the nearest RSU. However, to ensure privacy, a broadcast request from the city council will not include personal information such as fleet or vehicle IDs. Personal parameters are extracted before leaving the vehicle. This means only parameters such as vehicle speed etc. can be used by the city to aid in improving urban mobility. To further address issues surrounding privacy and security, which are major concerns in V2I, Apache Nifi has numerous processors for encrypting data. On receipt, dataflows on the RSU can immediately decrypt the incoming data with specific processors. However, as privacy and security are not in the scope of this body of work, such processors were not implemented. Figure 56 shows an overview of the algorithm on the vehicle when receiving a request from the RSU. All following scenarios represent the implementation of the broadcast fleet ID FFFF via the PSC parameter. Here, we are using the hypothetical scenario of 200, 400, 600, 800, 1250, and 1500 vehicles. However, only 20% of the vehicles in the vicinity belong to fleets, hence, only those vehicles will communicate on the service channel provided by the WSA, which equates to 40, 80, 120, 160, 200, 250, and 300 vehicles respectively.

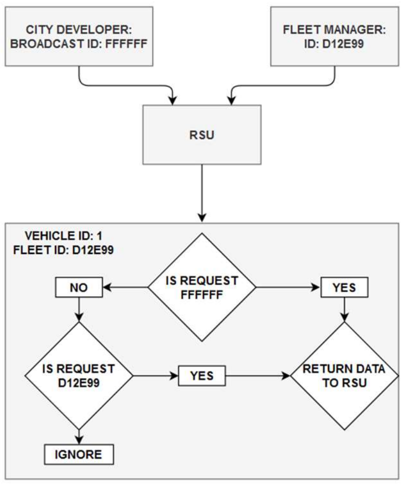


Figure 56: Logic performed on vehicle when receiving a request from RSU

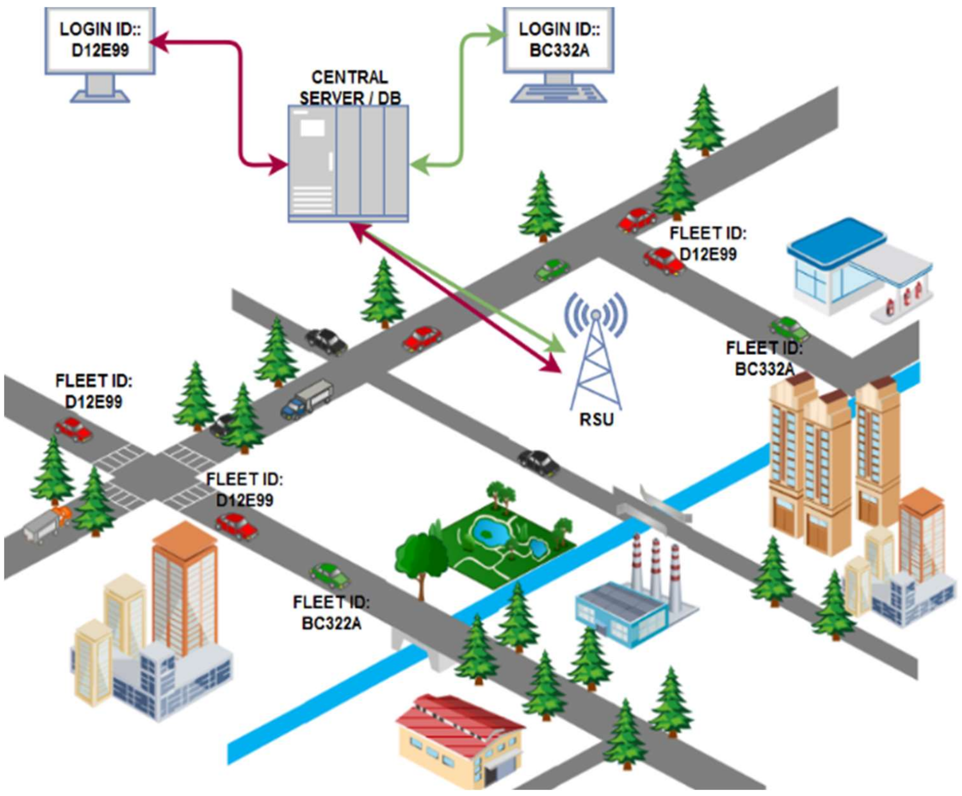


Figure 57: Multi-User Fleet Management use case

5.4 Use-Case II: RSU Governance

Previously we discussed adding a level of governance to the RSU, allowing the RSU to determine what lower layer parameters are configured in the vehicle to transmit messages. Utilizing the PSC parameter allows the RSU to pass extra control information to all vehicles in the area. We have created a format that allows us to pass multiple parameter changes which the vehicles are configured to adapt and modify the lower layers.

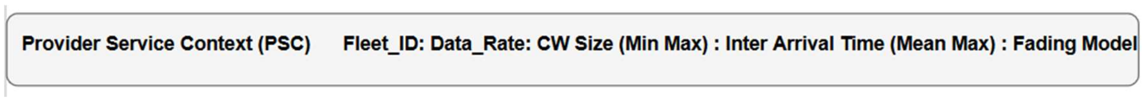

Provider Service Context (PSC) **Fleet_ID: Data_Rate: CW Size (Min Max) : Inter Arrival Time (Mean Max) : Fading Model**

Figure 58: Novel PSC Format for passing control info to vehicle

Figure 58 shows the format of our novel PSC layout with each section separated by a colon. In our novel PSC format, the Fleet ID parameter is controlled by the Cloud, however, all other parameters are added by the RSU and determined by current conditions. Below, we explain each section.

Fleet_ID – This parameter can be used by individual fleet managers to pass information to their own vehicles. Fleet ID FFFF is a broadcast for all fleets in the area to return information.

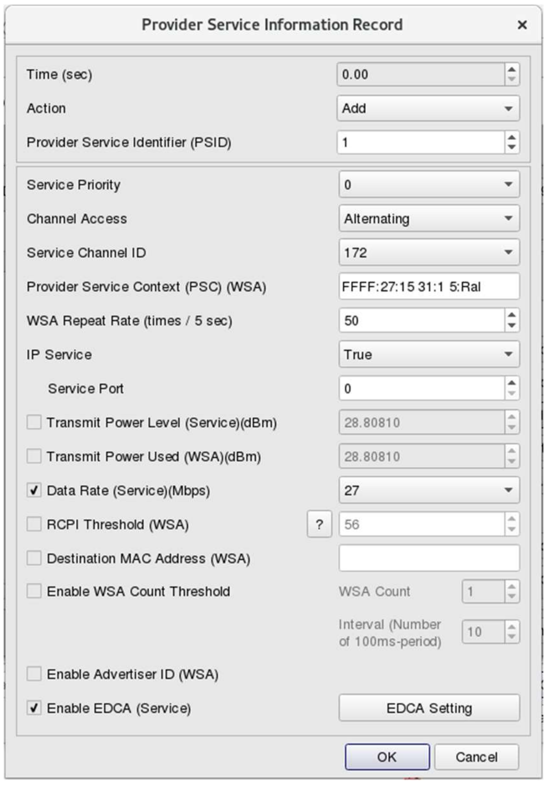
Data_Rate – Determines the preferred data rate for the vehicle to exchange information with the RSU.

CW Size (Min Max) – Passes the preferred min and max contention window size to the vehicles

Inter Arrival Time (Mean Max) – Modifies the transmit time that the vehicle sends information to the RSU. The mean and max parameter will be passed into an exponential distribution algorithm on the vehicle.

Fading Model – Determines which fading model the vehicle will use to transmit information (Raleigh / Ricean)

Figure 59 is an example of a WSA showing a configured PSC section using our format. The RSU may determine the parameters based on historical trends. Once the OBU receives this WSA, it extracts the PSC information and passes the parameters to the lower layers. In our example, we use a custom AWK script combined with FBP dataflows on the vehicle to filter out the parameters and pass them to lower layer configuration files via Apache MiNifi.



Time (sec)	0.00
Action	Add
Provider Service Identifier (PSID)	1
Service Priority	0
Channel Access	Alternating
Service Channel ID	172
Provider Service Context (PSC) (WSA)	FFFF:27:15 31:1 5:Ral
WSA Repeat Rate (times / 5 sec)	50
IP Service	True
Service Port	0
☐ Transmit Power Level (Service)(dBm)	28.80810
☐ Transmit Power Used (WSA)(dBm)	28.80810
✓ Data Rate (Service)(Mbps)	27
☐ RCPI Threshold (WSA) ?	56
☐ Destination MAC Address (WSA)	
☐ Enable WSA Count Threshold	WSA Count 1
	Interval (Number of 100ms-period) 10
☐ Enable Advertiser ID (WSA)	
✓ Enable EDCA (Service)	EDCA Setting

Figure 59: WSA with advanced PSC configuration

5.5 W-V6 Mechanism

WSMP is primarily used for one hop messages, whereas IPv6 can be used for multi-hop. When a higher layer application wants WSMs to be sent on its behalf, it sends a WSM-WaveShortMessage.request to the Wave Management Entity (WME). On success, the message is packaged as WSM and passed to lower layers for transmission. The WME keeps a Management Information Base (MIB) of current system and network information such as WSM max length, mac addresses, and Provider/User roles of the node and so on.

W-V6 is a novel mechanism that polls the WME before each message transmission. This is achieved via Apache MiNifi and UDP ports. The default UDP port for higher layer applications to communicate with WSMP is port 5000, Figure 61. If wsm.request = fail, the AMS is passed to a FBP dataflow that transmits via UDP. WSM requests may fail for a number of reasons, most common being the message is larger than the maximum WSM message size threshold in the MIB, Figure 61. In our scenario, wsm.request fails when the device role is 0 (not a provider or user). This effective mechanism means the OBU can continue to transmit its messages to the RSU via the RSUs IP address when it is out of range of the RSU. In our scenario, when WSM is not available, the message is transmitted using the vehicles 802.11n stack, which scans for open Wi-Fi hotspots. Figure 60 shows the W-V6 mechanism located in the application layer.

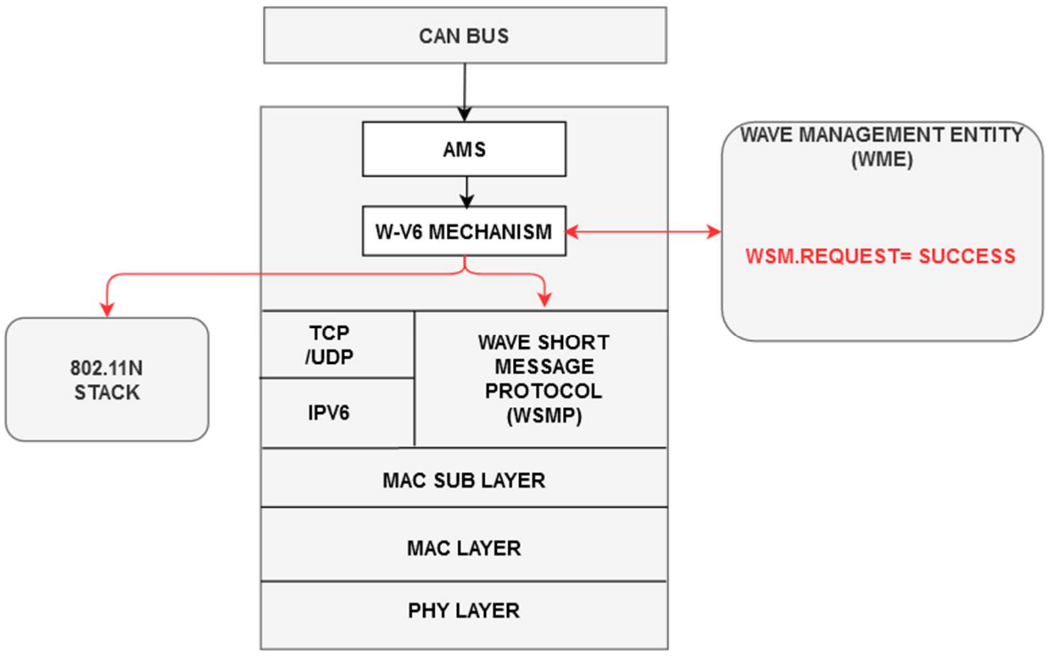
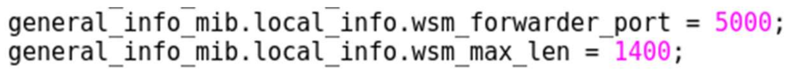


Figure 60: W-V6 Mechanism Operating in the Application Layer of OBU



```
general_info_mib.local_info.wsm_forwarder_port = 5000;
general_info_mib.local_info.wsm_max_len = 1400;
```

Figure 61: Snippet of WME configuration file

5.6 Evaluation Metrics

To evaluate WAVE-Flow, we measure the number of packet collisions, throughput and packet loss ratio (PLR). Collisions occur when two nodes transmit at the same time without sensing each other. Throughput is the successful message delivery over a communication channel and is measured in bits per second. Packet loss refers to the number of received packets versus that of sent packets. The packet loss ratio can be calculated by:

(Total Packet Transmitted – Total Packet Delivered) /

Total Packet Transmitted

Modelling driver behaviour is a complex but necessary responsibility within vehicle simulation since not all drivers behave and react the same way. In our simulations we defined a set of driver types and randomly distributed each type among the vehicles configuration files. The driver model determines the driver's acceleration, deceleration, and response to warnings. For this purpose, a classification of driver's characteristics was needed. As shown in Figure 62, drivers were grouped into three categories based on their desired speed [139]. Other behaviours such as lane changing, traffic light signals were also configured. Before simulations, a configuration file was created on the OBUs that utilized an STG (Source Traffic Generator) command that determines data granularity, packet size, communication protocol and destination address. On the RSU, a configuration file was implemented that utilizes a RTG command to ingest the incoming traffic, and where to log the results.

$$V_{desired} = \begin{cases} \text{Speed Limit -5} & \text{Conservative} \\ \text{Speed Limit} & \text{Normal} \\ \text{Speed Limit +5} & \text{Aggressive} \end{cases}$$

Figure 62: Three groups of driver types

5.7 WAVE-Flow Evaluation & Results

The layout of this section is as follows: First we perform an in-depth comparison of UDP and WSM, involving the modification of multiple lower layer parameters. We then demonstrate RSU governance over nearby vehicles. Finally, we perform simulations that implement WAVE-Flow and its mechanisms on the RSU and OBUs.

5.7.1 WSM and UDP Comparison

Here we perform a comparison of message types available in 802.11p. We also investigate changing lower layer parameters to improve both UDP and WSM in unicast scenarios, where the vehicle is transmitting to a RSU. Although it is widely known that WAVE supports WSMP and IPv6, to the best of this author's knowledge, there is no previous literature that evaluates and compares both protocols side by side. With the expectation that WSM will be more suitable in one hop scenarios, the following section will evaluate WSM and UDP messages of equal payload and matching lower layer parameters. The goal is to provide valuable insight into the packet loss ratio, number of collisions and throughput when comparing both message types in similar scenarios, and if we can enhance performance.

Secondly, results from this section will provide insight into the relevance of parameter changing in V2I. In a real-world scenario, RSUs will be static and have historical insights in traffic behaviour, allowing them to provide the vehicles with the most efficient parameters to transmit at a given time. This section aims to show the reader the impact certain parameters can make in unicasting scenarios.

We begin with simulations using the default WAVE parameters as described in Table 25. Then, we introduce the RTS/CTS mechanism and evaluate its inclusion. We also evaluate a range of available data rates for our messages. Finally, we evaluate different EDCA parameters and channel fading models. It must also be noted that we compare constant packet inter-arrival times with an exponential distribution inter-arrival time. Exponential distribution deals with the time between occurrences of successive events as time flows by continuously. Instead of vehicles sending data to the RSU in a continuous uniform fashion, for example, every 1 second, this algorithm uses three variables; min, max and mean time. In this example min is 0.1 second, max is 5 seconds, with the mean being 1 second. The goal of using this algorithm is to distribute the network traffic between the min and max time period, which in turn, reduces packet loss and improves throughput at the RSU.

During testing, a strenuous amount of simulations evaluating the modification of default parameters took place as presented in Table 25. However, to prevent repetitiveness in our results, we present the best performing combinations. In total 196 different simulations were performed as shown in Table 26, each simulation was run 3 times, taking the average as the presented results.

Table 25: Default Simulation Parameters

Grid Size (4*4)	1km x1km
Simulation Time	120 Seconds
Nodes	40,80,120,160,200,250,300
Data Rate	6Mbps
Transmission Power	28.8 dBm
Packet Size	100 Bytes
Path Loss Model	Two Ray Ground
Fading Model	None
Vehicle Max Speed	23 m/sec
Max Acceleration / Deceleration	1.5 m/sec^2 / 4 m/sec^2
Packet Inter-Arrival Time	1 sec constant

Table 26: Breakdown of simulations evaluated for WSM and UDP

Number of Simulations Performed for WSM and UDP	Total
2 Inter-Arrival Times * 7 (Different Numbers of Nodes) * 2 (UDP + WSM)	28
5 Data Rates * 7 * 2	70
Inclusion of RTS/CTS * 7 * 2	14
4 EDCA Settings * 7 * 2	56
Raleigh / Ricean Fading Models * 7 * 2	28

5.7.2 *Default Parameters of WSM and UDP*

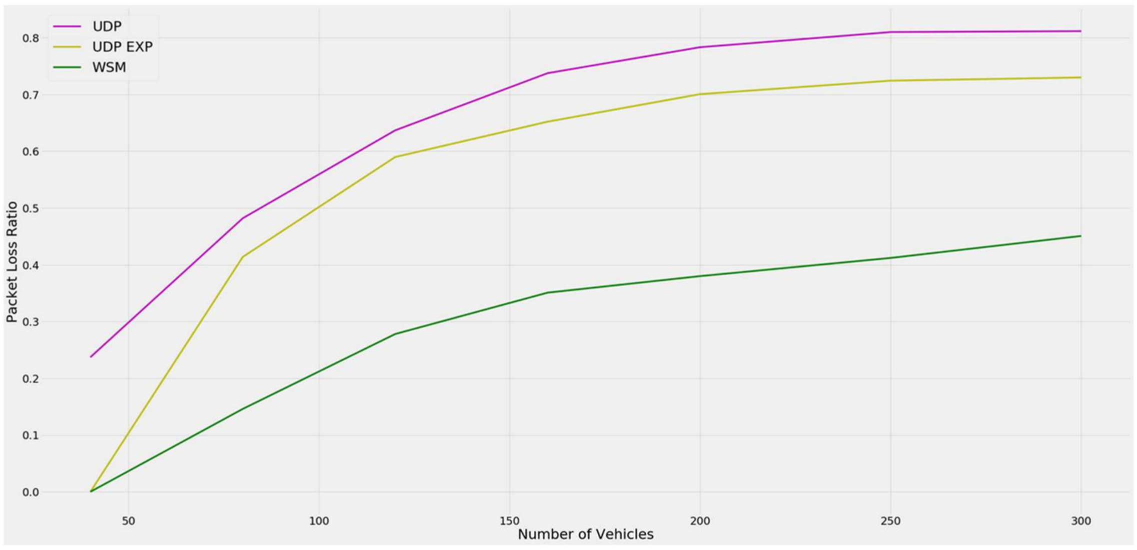


Figure 63: WSM and UDP (UDP with constant and exponential inter-arrival times)

As shown in Figure 63 UDP suffers greatly with the increase in nodes, with a packet loss as high as 0.81. However, the inclusion of an exponential distribution algorithm for inter-arrival packets times improves UDP to 0.73. WSM performs much better with PLR as low as 0.44. From here on, we will exclude UDP constant as it performs very poorly. The following graphs are collisions and throughput in the same scenario.

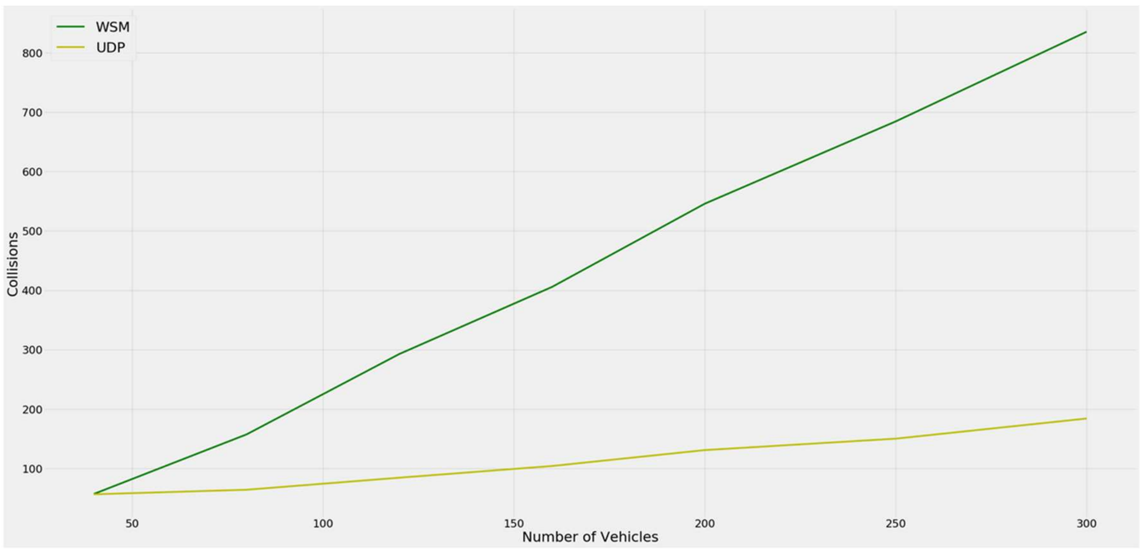


Figure 64: Average No. of Collisions per second for WSM and UDP

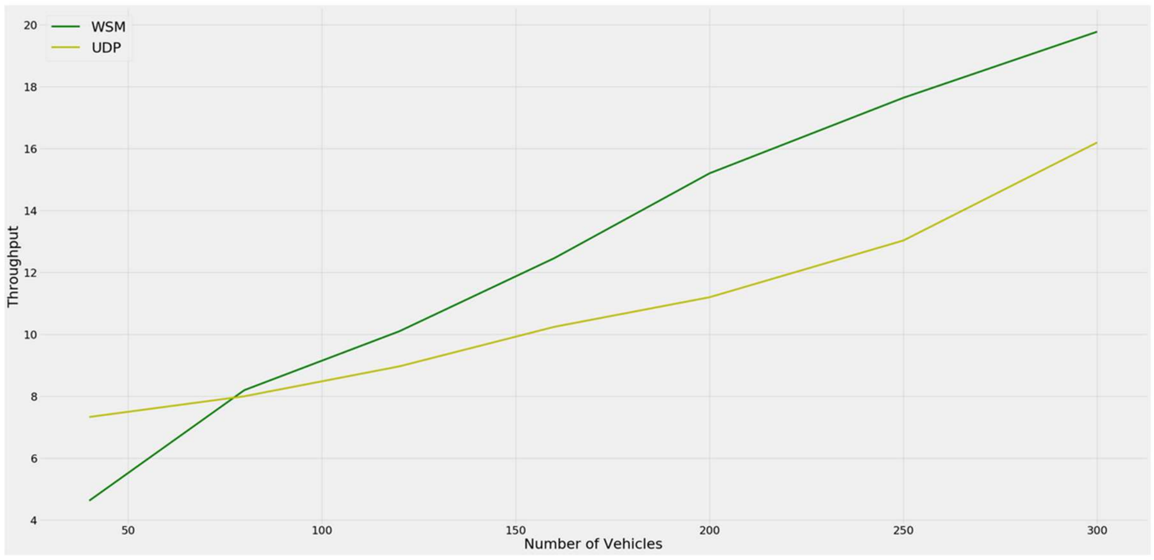


Figure 65: Average throughput in Kbps for WSM and UDP

5.7.3 RTS/CTS and Data Rate

In this section we introduce RTS/CTS. As use of RTS/CTS can potentially reduce throughput because of acknowledgment packets etc., a threshold value is used. In most routers or devices, the default threshold is 2500 bytes. The default threshold was 3000 bytes in the EstiNet simulator. This means that RTS/CTS won’t be activated unless a packet size is greater than the threshold. However, for our testing, we lowered the threshold to 50 bytes, allowing our messages to activate RTS/CTS mechanism.

We also present in our graphs the transmission of packets at the default 6Mbps and the highest available data rate 27Mbps. It must be noted, 3Mbps, 12Mbps, and 18Mbps were also tested but varied little when comparing to 6Mbps, so are excluded. Using the 27Mbps data rate requires a higher transmission power. As previously stated, this is not recommended in broadcast scenarios, as it brings a higher amount of interference from other vehicles. However, as we are unicasting to a specific node, this should not have a negative impact. In this section, WSM and UDP will be graphed separately. The three lines represent WSM with default parameters, WSM with RTS/CTS activated at the 6Mbps data rate, and also the 27Mbps data rate.

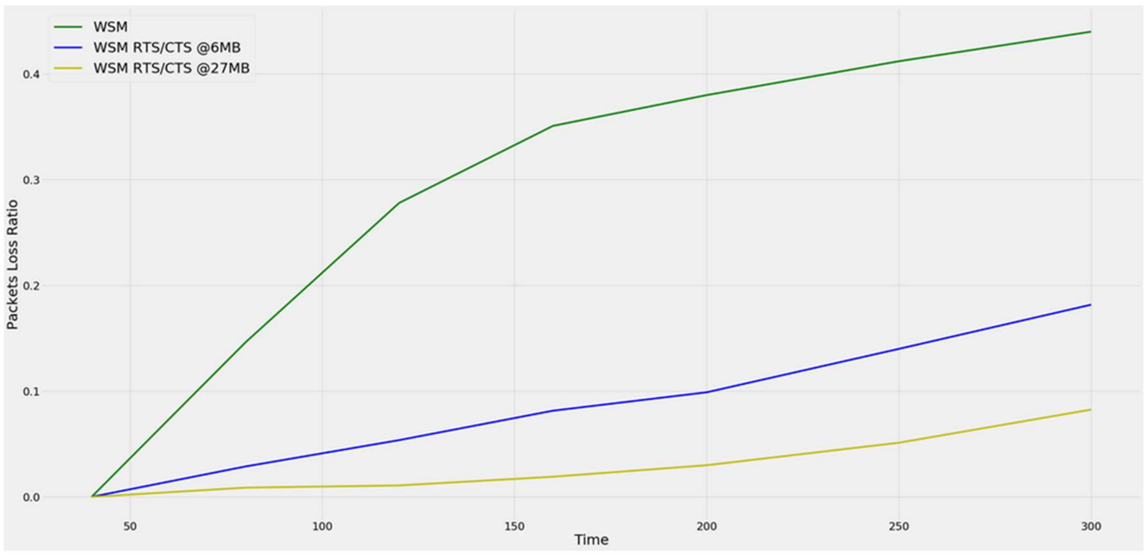


Figure 66: WSM PLR with Inclusion of RTS/CTS and Higher Data Rate

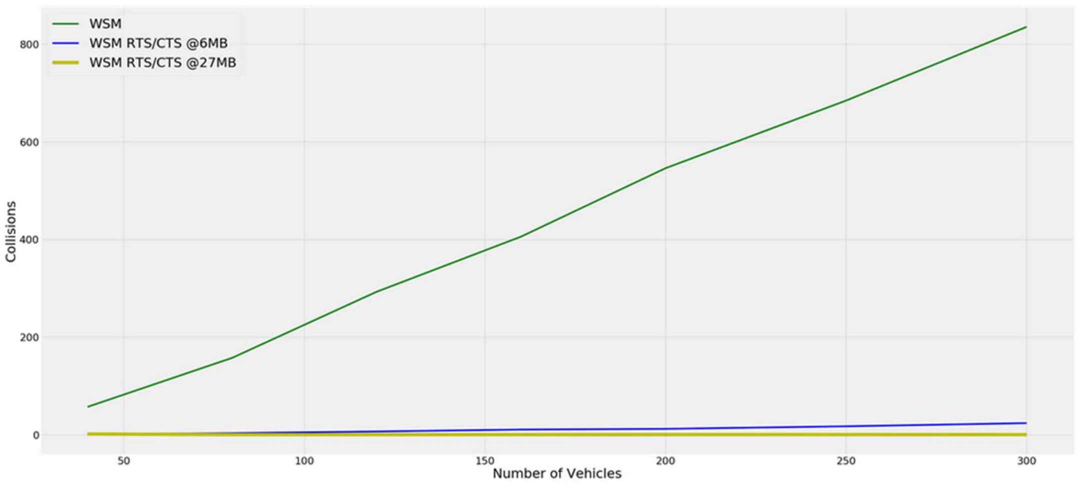


Figure 67: WSM Collisions with Inclusion of RTS/CTS and Higher Data Rate

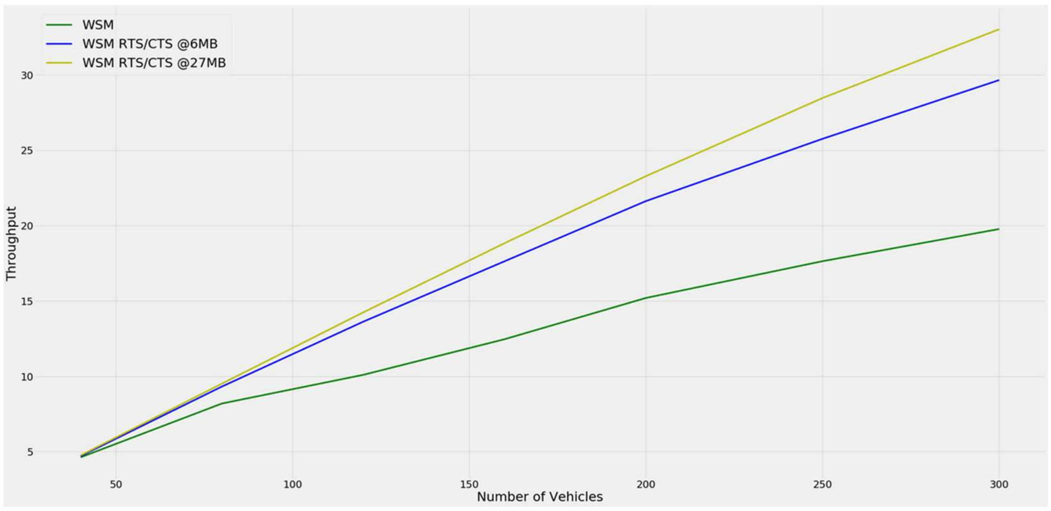


Figure 68: WSM Throughput with inclusion of RTS/CTS and Higher Data Rate

As can be seen, RTS/CTS adds a significant increase in performance for WSM. PLR drops from 0.44 to 0.17 with the inclusion of RTS/CTS. This is further improved when data rate is raised to 27Mbps. In total, the inclusion of higher data rate and RTS/CTS improves PLR from 0.44 to 0.08, a reduction of 0.36. This is also visible in throughput, with a total increase 16%. What is striking, is the amount of collisions that occur in WSM, and the improvement RTS/CTS make when implemented.

As will be shown in the next section, collisions in WSM are much higher than UDP, even though packet loss is lower in WSM. The high collisions may be due to the quick packaging and retransmission attempts of WSM packets when compared to the slower IP protocol. The following section evaluates UDP in the same scenarios. Similarly, the three lines represent UDP with default parameters (now using the exponential distribution), UDP with RTS/CTS implemented at 6Mbps and also 27Mbps.

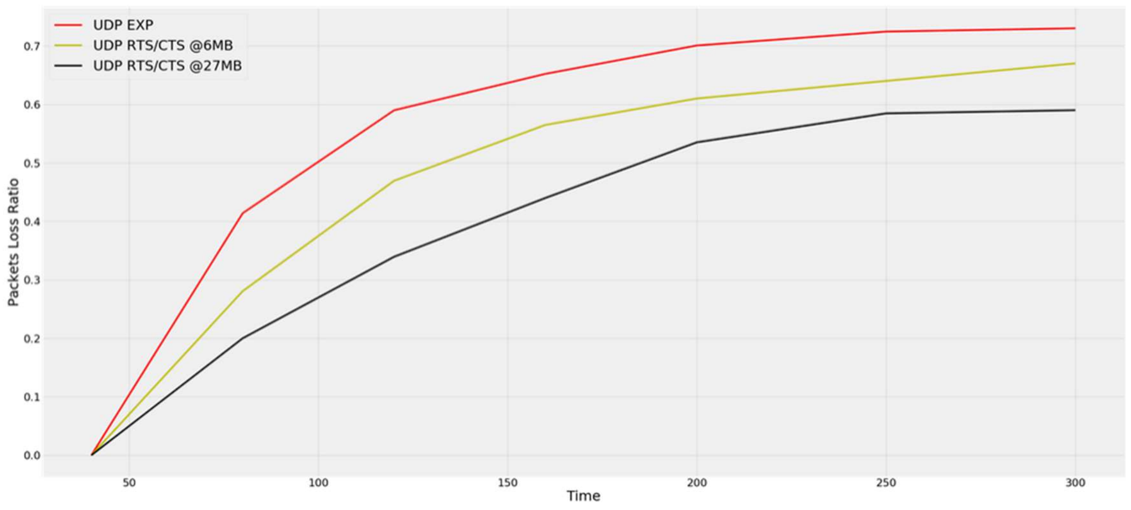


Figure 69: UDP PLR with inclusion of RTS/CTS and Higher Data Rate

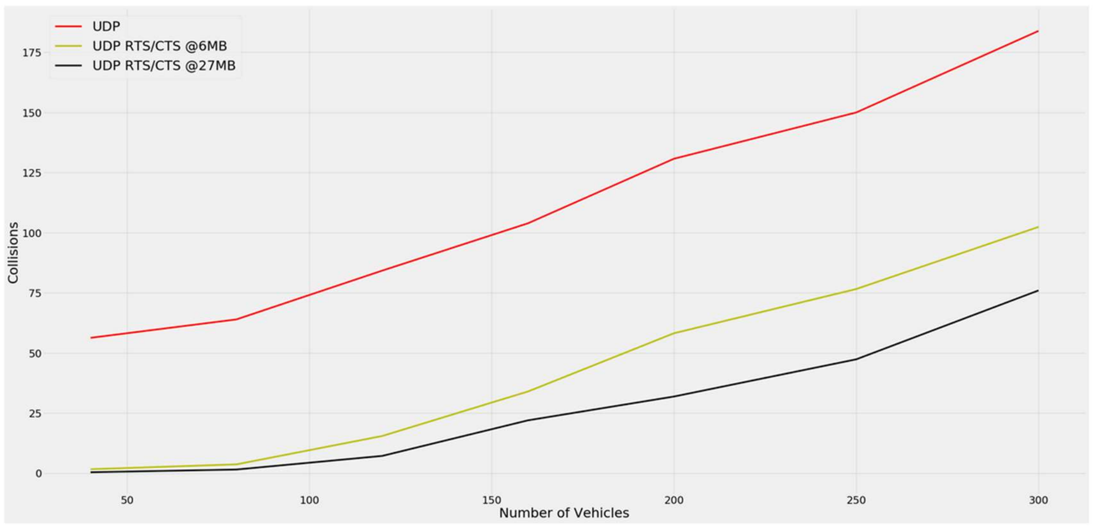


Figure 70: UDP Collisions with Inclusion of RTS/CTS and Higher Data Rate

It is visible that although RTS/CTS and a higher data rate improve UDP, the improvement is not as impressive as WSM. Packet loss improves from 0.73 to 0.59. A noticeable difference again is the reduction in collisions.

5.7.4 EDCA Settings & Fading Model

Next we modify the EDCA parameters and channel fading models. Up to now, the default EDCA Access Category was used, which was Best Effort (BE). As shown in Table 2 in the literature review, the Contention Window for BE is 15, meaning a back-off time of up to 15ms occurs before a packet can retransmit. However, as results will show, it is not always the best performing CW. Finally, we apply the two most common fading models, Raleigh and Ricean, as the default settings for fading models are empty in EstiNet. The following graphs show the PLR, throughput and collisions from our final combination of lower layer parameters, when compared to the initial default parameters, and results from the previous section.

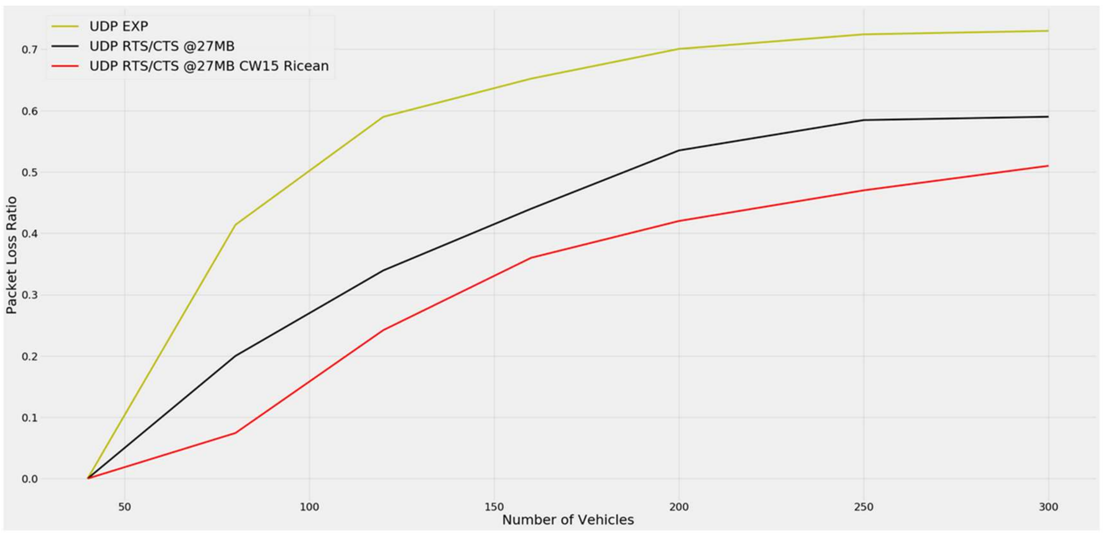


Figure 71: Default UDP Vs Final UDP Parameters.

As shown in Figure 71, for UDP, the CW of 15 performed best out of the four EDCA default parameters. Also, the Ricean fading model slightly outperformed the Raleigh model. Overall the modification of both parameters improve on our previous efforts from 0.59 to 0.51. In the following graphs we see a similar trend with a further reduction on collisions and increase in throughput. For three hundred vehicles, collisions per second dropped from 184 to 53, whereas throughput increased from 15.6Kbps to 21.8Kbps.

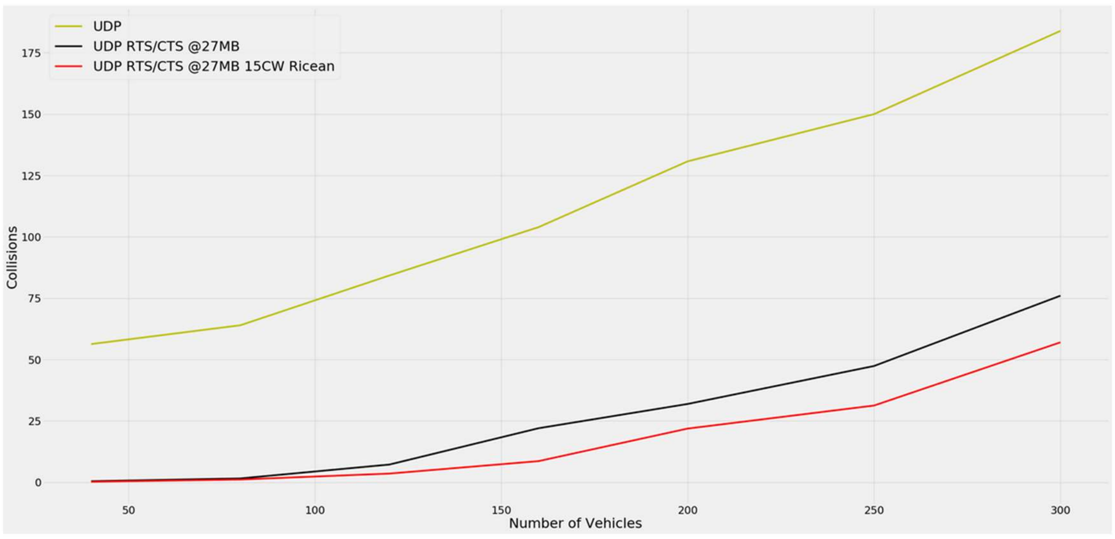


Figure 72: Collisions for Default UDP Vs Final UDP Parameters

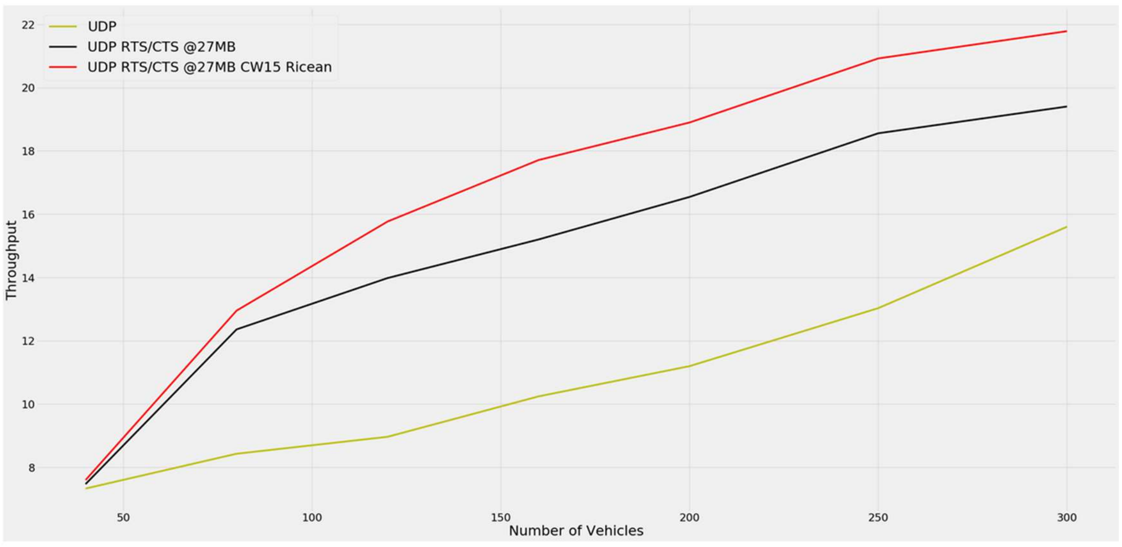


Figure 73: Throughput for Default UDP Vs Final UDP Parameters

As visible in Figure 74, our final set of parameters make a significant reduction in packet loss for WSM, reducing initial PLR of 0.44 to a minimal 0.014. In terms of EDCA, CW of 31 performed best for WSM as opposed to CW 15 for UDP. The Ricean model also slightly outperformed the Raleigh model, while slightly improving overall performance. Figure 75 shows the collisions in WSM. The introduction RTS/CTS brought collisions close to zero and is hard to distinguish between the lines in the graph. As mentioned previously, WSM suffers from high collisions without RTS/CTS, with over

800 collisions per second with 300 vehicles. The final set of parameters reduced this number to below 1. Throughput, Figure 76, increased from 19.77Kbps to 36.2 Kbps. Table 27 provides visual representation of the final lower parameters for both UDP and WSM.

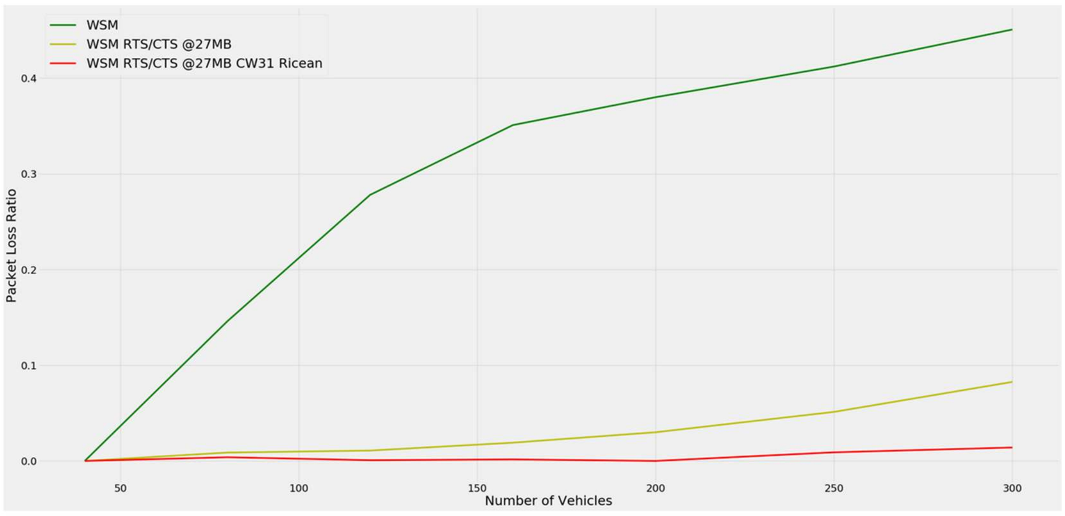


Figure 74: Packet Loss Default WSM Parameters Vs Final Parameters

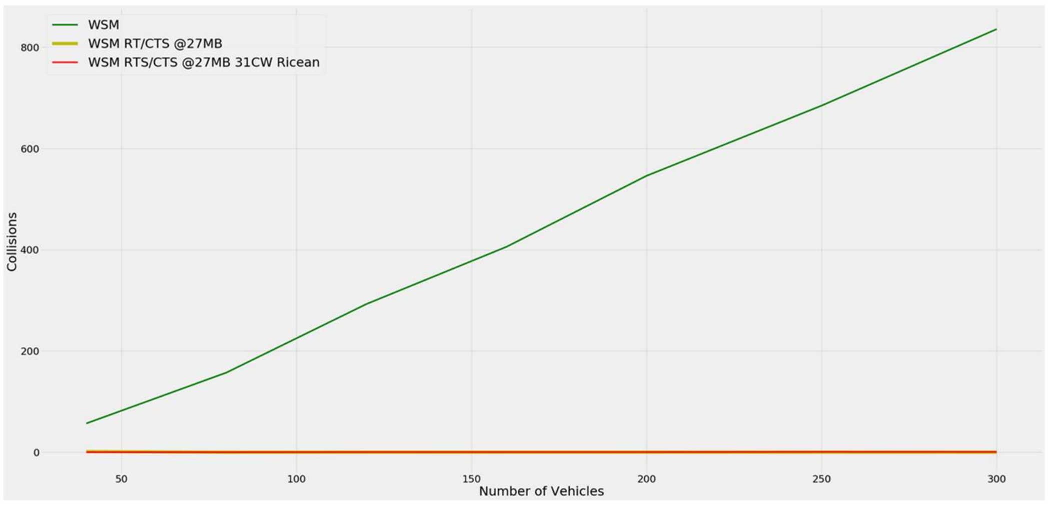


Figure 75: Collisions for Default WSM Vs Final WSM Parameters

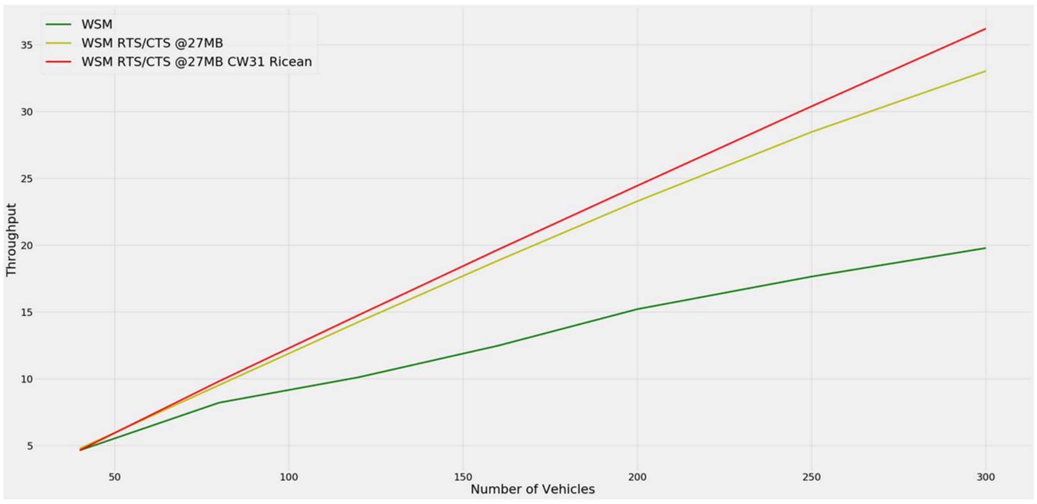


Figure 76: Throughput for Default WSM Vs Final WSM Parameters

Table 27: Final Lower Layer Parameters

Data Rate	**27Mbps**
Transmission Power	28.8dBm
RTS/CTS	Activated
Contention Window Size	15 (UDP) 31 (WSM)
Fading Model	Ricean
Packet Inter-Arrival Time	Exponential Distribution

5.7.5 Summary

Results show WSM to be a far superior message type in one-hop scenarios. However, with modification of multiple parameters, we improved PLR in both packet types. WSM packet loss was decreased from 0.44 to 0.014 for 300 nodes. This is a significant decrease of 0.426. As packet delivery to the RSU is the inverse of this, it means packets received by the RSU increased by 42.6%. There were also performance increase in UDP, improving packet loss from 0.73 to 0.51, a decrease of 0.22 for 300 vehicles. Overall, IPv6 is still important as it is relevant for internet purposes or its routing ability, so it is in our interest to continue to evaluate it in our use case.

5.7.6 RSU Governance

As a use case for WAVE-Flow we discussed RSU acting as governing bodies over Connected Vehicles. Here, the RSU determines the granularity at which messages are transmitted based on the number of nodes in the area. To do this, we need to perform multiple simulations consisting of different numbers of nodes transmitting at different granularity. This will allow us to build an algorithm to implement on the RSU. The following graph represents a total of 28 different simulations, each run 3 times, using the average.

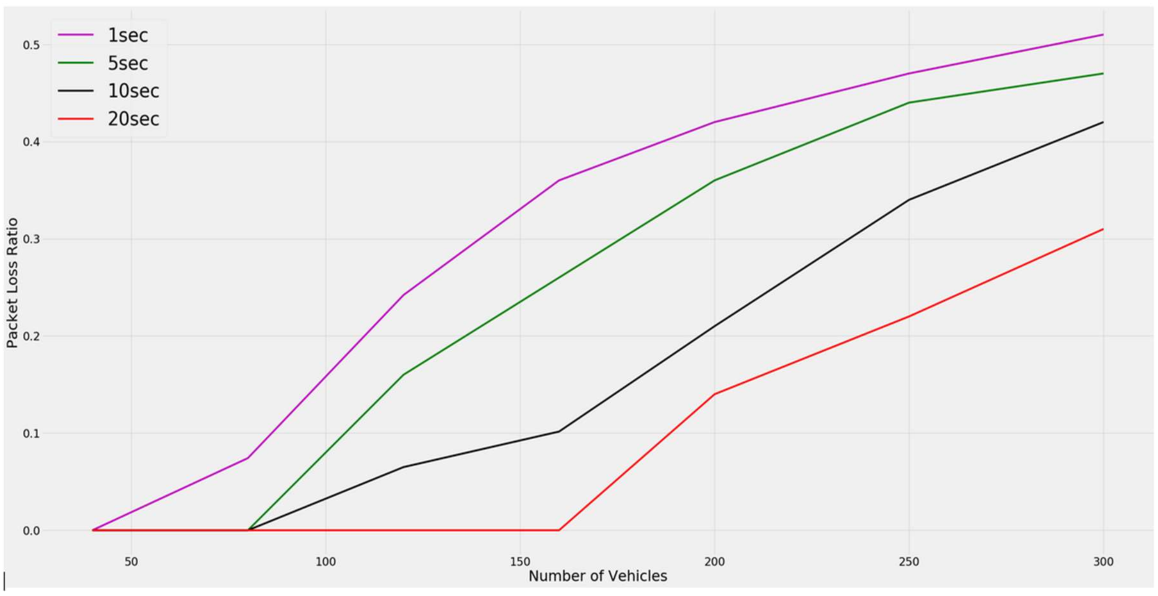


Figure 77: PLR of different transmission times

For this example we use UDP packets. Using the information presented in the graph, the RSU can create an algorithm that will modify the granularity of the vehicles packet transmissions as the number of nodes grow. An example of the pseudocode to keep packet loss as low as possible while data granularity high as possible, would be as follows:

```
If $V_n < 80$
    $T_I = 1s$
If $80 < V_n > 120$
    $T_I = 5S$
If $120 < V_n > 200$
    $T_I = 10S$
Else $T_I = 20s$
```

Where V_n is number of vehicles and T_I is Transmission Interval. In a real world scenario, a more self-adapting algorithm may be created from historical data. However, in this example the aim is to demonstrate a proof of concept. With this algorithm implemented, the RSU can then govern the rate at which Connected Vehicles transmit their data. What this has shown, and building on the previous section, is through modification of lower layer parameters and reducing data granularity, the RSU can pass control information to the vehicles that reduces overall packet loss from 0.73 (default UDP parameters previously discussed) to 0.31(modified parameters transmitting at 20 seconds).

5.7.7 WAVE-Flow Simulation

This final set of simulations aims to implement the previously mentioned methods and logic. In the first scenario, only vehicles within range of the RSU can communicate as shown in Figure 79. In the second simulation, we add three 802.11n access points (Wi-Fi hotspots) in areas outside the range of the RSU. We also implement W-V6 on the vehicles with the following logic applied.

```
If WSM. Request = Success
Then, port 5000 (WSM)
Else, FBP Flow to 802.11n
Scan for open Wi-Fi
```

To perform governance, the previously created data granularity algorithm is implemented on the RSU, which allows the RSU to determine the granularity of the vehicle transmission times. The earlier defined lower layer parameters for message transmission are implemented on the vehicles. Table 28 shows the simulation parameters.

Table 28: Simulation Parameters

Grid Size	**2Km * 2KM**
Simulation Time	400 seconds
Number of Nodes	300
Communication Standard	802.11p / 802.11n
Message Type	WSM /UDP
Mechanisms	RSU: Data Granularity Algorithm OBU: W-V6

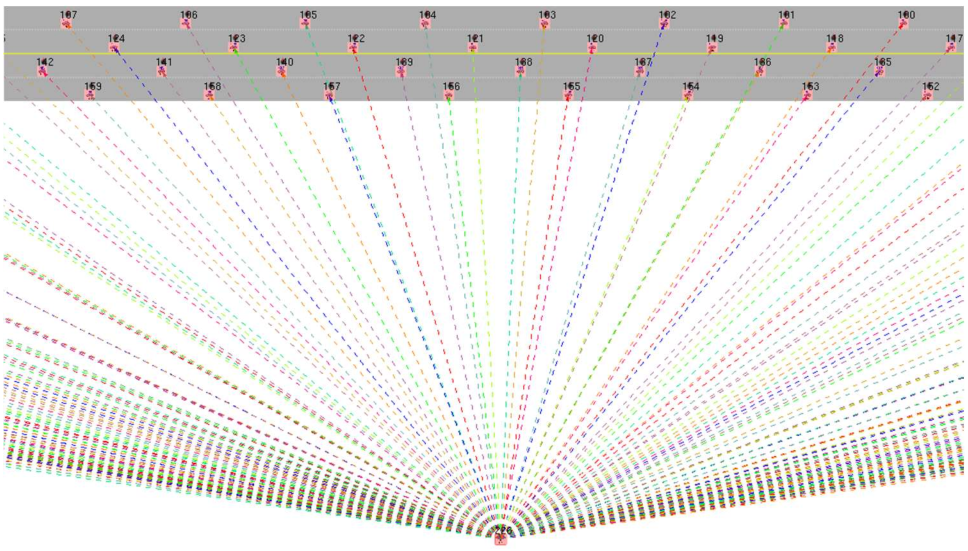


Figure 78: Close Up of RSU and OBU Connectivity

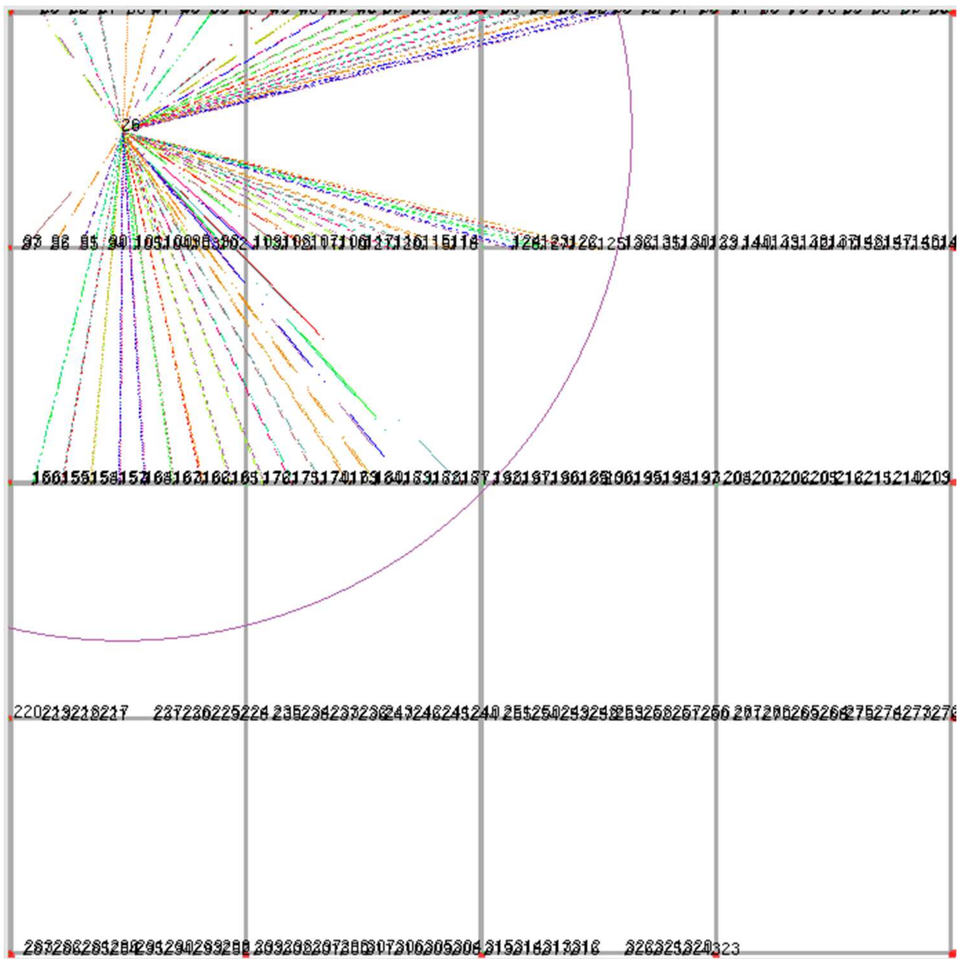


Figure 79: Simulation 1 Showing RSU Transmission Range (Circle)

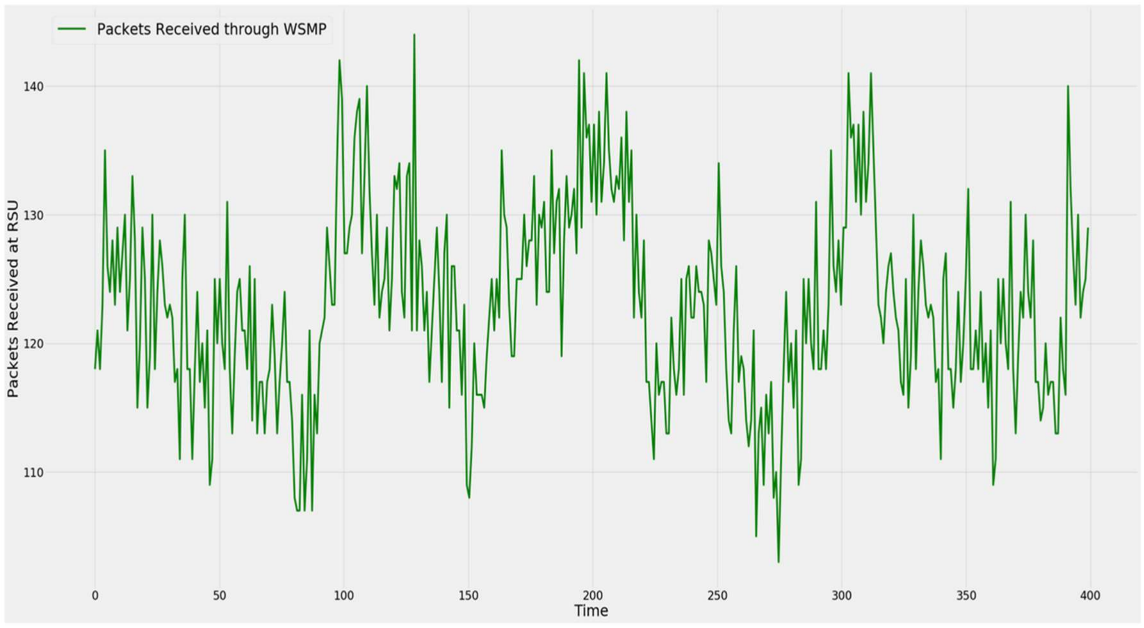


Figure 80: Simulation 1. WSM Packets Received at RSU

Figure 80 shows that although there are 300 nodes in the simulation, on average, 123 are within transmission range of the RSU. Other nodes outside the RSU range have no way of communicating. For this reason, we introduce three Wi-Fi hotspots, and implement W-V6 on the vehicles. As mentioned previously, once the WME denies the WSM request, the vehicles will send the packets to the 802.11n protocol stack. Which in turn, scans for an open Wi-Fi hotspot to transmit packets back to the RSU.

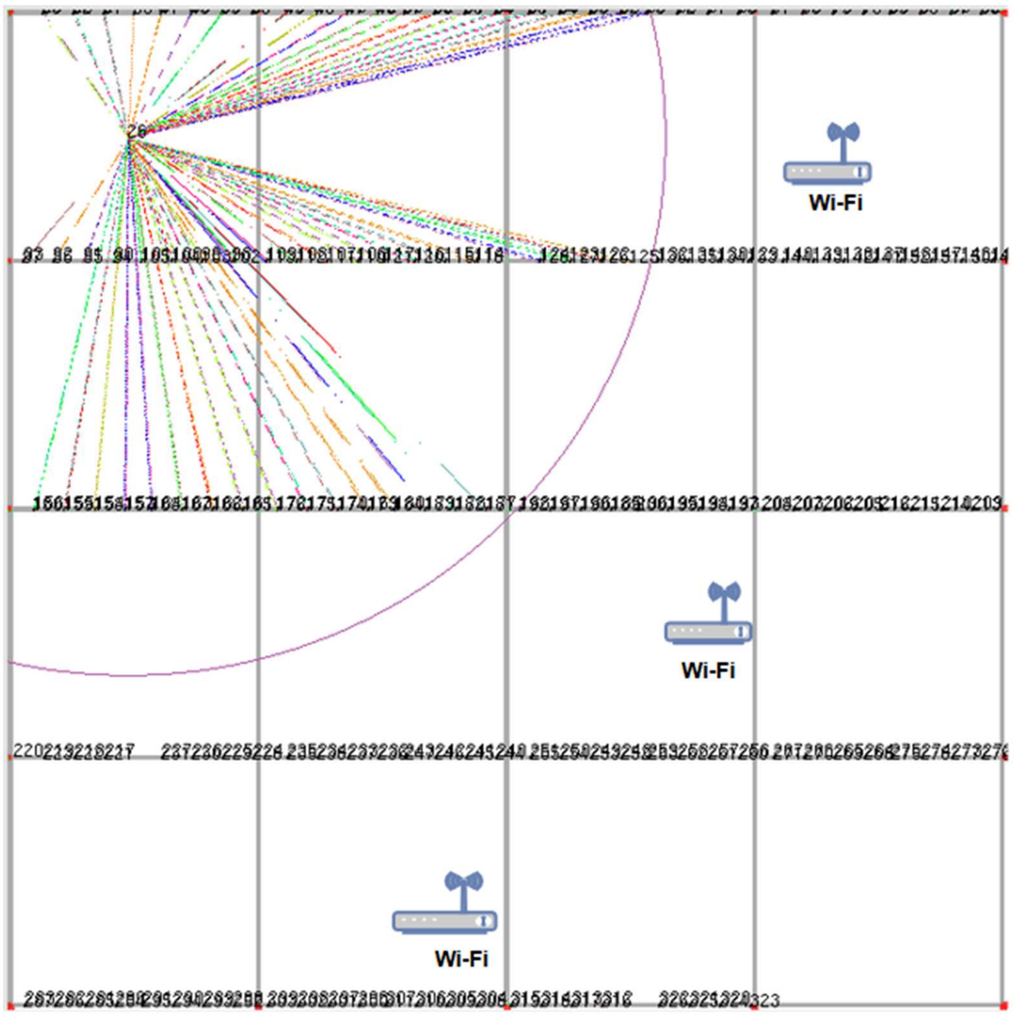


Figure 81: Simulation 2 with added Wi-Fi Hotspots

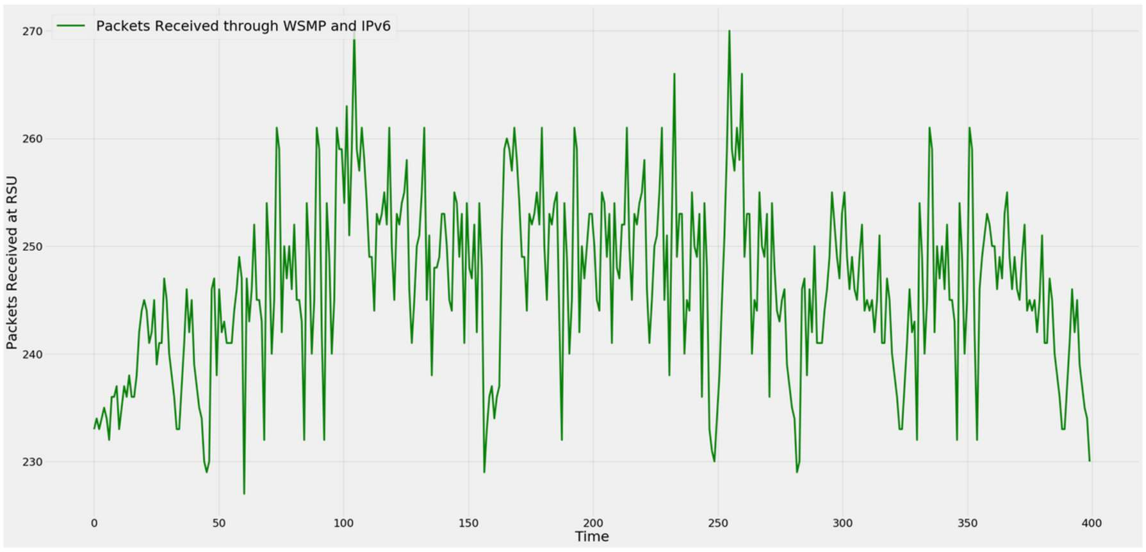


Figure 82: Packets received at the RSU via WSM and IPv6

As shown in Figure 82, vehicles that are out of range of the RSU, but in range of a Wi-Fi hotspot can still transmit packets to the RSU. The inclusion of W-V6 increased packet delivery at the RSU from 123 to 244 packets per second. Secondly, the RSU modified the packet transmission times of the RSU using the algorithm provided earlier. However, it is hard to distinguish its results when combined with incoming IPv6 packets. For this reason, Figure 83 shows the transmission of 1 vehicle of the course of the simulation.

In the following graph, we describe the 5 phases of transmission this vehicle goes through over the course of the simulation. The gaps between transmissions occur when the vehicle is outside the range of the RSU and Wi-Fi access points. Phase 1 shows this vehicle starting in the range of the RSU and transmitting messages every 1 second via the exponential distribution algorithm. This tells us that were less than 80 vehicles in range of the RSU at this time. However, while still in range of the RSU, the vehicle starts transmitting every 5 seconds in Phase 2, meaning there was between 80 and 120 vehicles in the area. There is a gap between transmissions until the vehicle comes within range of a Wi-Fi hotspot (Phrase 3), transmitting back to the RSU, still using the 5 second exponential distribution as this was the last configuration received from the RSU. Phrase

4 sees the vehicle back within range of the RSU, however, at this time, the vehicle is transmitting every 10 seconds, as per request of the RSU. This tells us at this time there were between 120 and 200 vehicles within range of the RSU. There is a lengthy gap until phase 5, until the vehicle connects to another Wi-Fi signal. However, it is still transmitting every 10 seconds as this was the last configuration it received from the RSU.

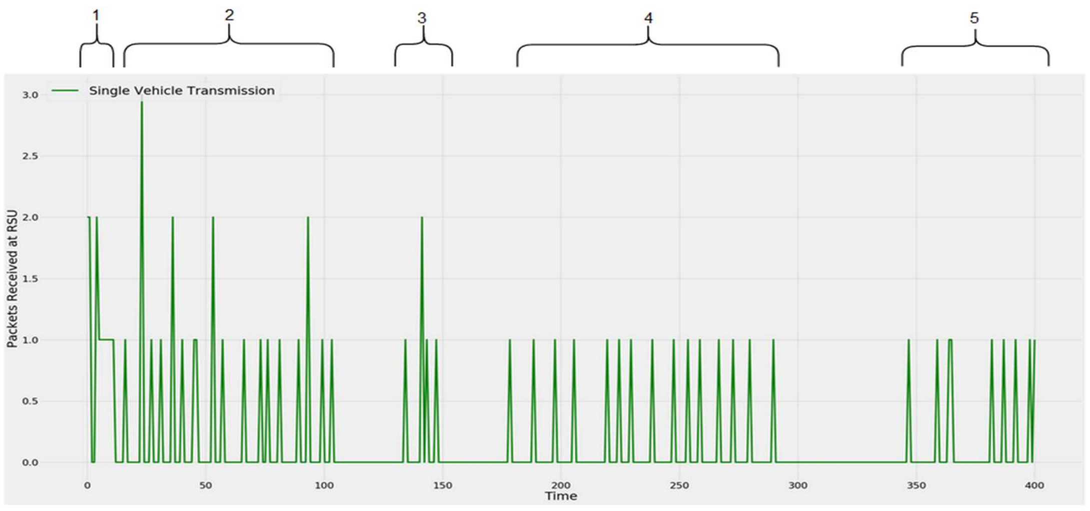


Figure 83: Phases of Single Vehicle Data Transmission over 400 Seconds

5.8 Conclusion

This section introduced WAVE-Flow and its mechanisms within WAVE nodes to enhance packet delivery and improve governance via the RSU. We also provided an in-depth comparison of WSM and UDP in one-hop scenarios, while paying special attention to the modification of lower layer parameters. This resulted in a combination of parameters that greatly improve packet loss and collisions for both protocol types. In our final simulation, we showed how W-V6 can enlarge the service area of an RSU by utilizing other medium in the area such as Wi-Fi hotspots. We also presented a case for RSU governance through utilizing the PSC parameter to pass control information to the Connected Vehicles.

6 Conclusion & Future Work

With IoT well and truly upon us, handling the influx of generated data is of paramount importance. The introduction of Edge and Fog-Computing, coinciding with Real-Time Processing technologies will greatly enhance the success of IoT. For this reason, we introduced our reference architecture with the goal of addressing the following research question.

Is it possible to design a data-centric architecture with the necessary performance to support bi-directional communication between Edge/Fog/Cloud nodes in an IoT paradigm?

However, in defining a reference architecture for IoT, certain requirements must be addressed, such as evaluating a suitable programming platform, local processing potential, data capturing techniques, coordination and governance capability. For this reason the following research questions were first addressed.

1) *In terms of communication and collaboration between nodes, to what degree can Flow Based Programming address the heterogeneity of services and applications in Vehicle-to-Infrastructure scenarios (V2I)?*

To address this, a Flow Based Programming architecture called DAGLADS was proposed. Implementations of DAGLADS showed its capability in bi-directional communication abilities, which is paramount for developers in IoT scenarios. Testing DAGLADS against the current state of the art centralized approach, Hortonworks DataFlow, we successfully emulated the advanced processing capabilities of the Cloud onto our Edge devices, minimizing data transmission in the range of 87% - 99% while maintaining computational accuracy. Further implementations were made to show how DAGLADS could perform advanced analytics such as real-time prediction through learning models. However, our overall goal was to introduce an architecture that spans across Edge/Fog/Cloud. For this, we needed a suitable use-case, Connected Vehicles in Smart Cities. However, this topic led to addressing the following questions.

2) *For vehicle telematics to be considered a key enabler in the development of Smart Cities, which in-vehicle data capturing technology provides the most accurate sensor data?*

Although future OBUs will ingest data directly from the CAN, currently most literature avails of CAN data through the OBDII port. A section of this thesis aimed to compare OBDII and FMS, a subset of CAN parameters for vehicle telematics. Results have shown OBDII to be unsatisfactory in a number of areas when compared to FMS. This body of work also gave us insight and the ability to create our own message set for vehicle telematics in a WAVE scenario. The goal of this work was to provide the research community with an evaluation and a comparative dataset showing OBDII may not be completely accurate when compared to FMS.

3) *Can a multi-tier Flow Based Programming architecture enhance the distribution and governance of vehicle sensor information in Smart Cities?*

Smart Cities and Connected Vehicles are prime examples of Edge/Fog/Cloud. For this reason, we developed an IoT application platform called WAVE-Flow, with the focus on V2I in Smart Cities. There is a wealth of literature discussing V2V and V2I, however, the majority of these works are communication based; i.e. improving routing protocols. With much mention of OBUs and RSUs, very little work exists in how these devices ingest, process and distribute vehicle information. Our goal was to evaluate how our FBP inspired architecture could compliment one of the more complicated areas of IoT. To do this, adjustments were made to our original DAGLADS architecture to allow for the incorporation of RSUs and 802.11p communication standard.

Through the creation of novel concepts such as utilizing the Provider Service Context parameter as a way of governance for RSUs, W-V6 for medium switching, our AMS message and the novel fleet id system, we have shown WAVE-Flow to be a viable platform for V2I. In-depth testing of some of the common parameters associated with V2V/V2I also provided the authors, and readers, with valuable insight into the performance of WSM and UDP packets under similar circumstances. It is our firm belief, each of the above have contributed to this field, and more importantly, created new avenues of research. PSC for example is a relatively unknown parameter that if implemented correctly may be useful in multiple scenarios where communication to specific subset of vehicles is required.

With future vehicles expected to consist of multiple communication interfaces, WAVE, 802.11, 4G, 5G, we evaluated a novel mechanism that switches between interfaces dependent on proximity to a RSU. Results have shown WSM to be a superior message format to distribute when close to a RSU, however, as WSM is a one hop technology, it does not scale well in terms of distance from RSUs. This requires a change in medium if the vehicle is still transmitting to the RSU.

RSUs will be static devices with access to much historical information of a specific area in a city. Self-adapting algorithms created from this information may be used to adjust lower layer parameters on vehicles, improving overall communication in V2I and V2V. In this body of work, we provided a novel concept in which the RSU controls the lower layers of the OBUs, along with a proof of concept use case.

Through implementations of our reference architecture, we have shown how governance over processing data at source can be achieved by nodes throughout the Edge/Fog/Cloud.

1) The Edge node via internal parameters such as CPU usage, network connectivity etc.

2) The Fog nodes (RSUs).

3) The Cloud via control information from the developer.

6.1 Limitations

This body of work spanned across a broad area, combining Edge/Fog processing, CAN signals, vehicle telematics before focusing on V2I. Although a thoroughly enjoyable experience, time was of the essence. V2V/V2I is a complicated field of study with multiple areas of research required. For example, modifying lower layer parameters to enhance packet delivery opens up a lot of new research questions and literature. QoE mechanisms, data rates, propagation and channel fading models were all mentioned and tested in this body of work. However, each in itself are a large area and out of scope of this work. The author would consider this a limitation of this work as to not solely focus on one or two of these areas for maximum performance of specific parameters. However, as the focus of this work was the architecture, and modifying these parameters were merely a use case to show how a RSU could adjust OBU parameters, it is our admission that more expertise is needed in each area for future work.

The choice of VANET simulators is not an easy one. Each simulation tools have pros and cons. Veins, the most widely used VANET simulation tool and with a wealth of online support, does not support unicast in 802.11p. For this reason, EstiNet was chosen. EstiNet provides a platform to get up and running quickly as it has a strong GUI base. This was very helpful as creating VANET scenarios are renowned to be a difficult achievement. However, EstiNet also has some cons. Lack of support being the main one. Its predecessor NCTUNs was open source with much more support in previous years. However, EstiNet is now a paid for service with priority code and minimum online support groups. This made debugging of small issues very time consuming. Another negative aspect of EstiNet is the time it takes for a simulation. Running a simulation of 300 nodes can take up to 7 hours. This could be due to the fact it uses your devices IP stack. Its lack of support for routing protocols is also problematic. Previous versions of Estinet supported AODV and DSR, however, that is no longer the case.

6.2 Future Work

As mentioned in the limitations section, this body of work spanned quite a large area, opening doors for future work in many areas. Fog-Computing has potential to impact a wide range of requirements in various applications including Smart Cities, healthcare, transportation, and large-scale industries. However, currently there are limitations in regards to platforms for developers to deploy and execute generic IoT applications on Fog devices. Connected vehicles are in a unique position in IoT as they have potential to act as Fog nodes due to their large processing and storage capability. Connected Vehicles collaborating as Fog devices is a research area that is gaining momentum. Further collaboration between vehicles and RSUs to perform advanced processing capabilities and storage is an area WAVE-Flow could expand to.

In the Fog-Computing paradigm, there are many unanswered questions. How to determine which task or services should be processed locally or offloaded, and where best to offload to, are ongoing research questions. Offloading and distributing IoT applications will play an important role in IoT, however, the advantages of offloading regarding the delay, bandwidth, and energy consumption is still an open research issue. If the predicted performance does not gain any significant advantages, then offloading is not beneficial.

Future work would aim to address the above issues with the following objectives.

Enhance the WAVE-Flow platform with self-adapting configuration capabilities, through the use of learning algorithms. This will allow for an evaluation of various parameters that have an impact on the offloading performance of an application, such as energy consumption, caching, bandwidth cost and latency.

Create a distributed environment where that in the event of offloading a task is required, an evaluation can be provided on efficiently offloading applications to the most suitable nodes in the network.

As discussed throughout this work, Fog-computing will play a significant role in two IoT scenarios in particular, Smart Cities and Connected Vehicles. Future work could consider a use case in which data generated from Connected Vehicles' camera feeds can enhance urban surveillance in a Smart City. Tasks such as traffic surveillance, face detection and riot prediction may be collaborated and distributed between the Connected Vehicles and RSUs throughout the city. This would require a group of vehicles in close proximity to distribute the processing of live video feed among each other, with the capability of offloading the heavier work load to the RSU, or Cloud.

References

[1] J. Pan and J. McElhannon, ‘Future Edge Cloud and Edge Computing for Internet of Things Applications’, *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 439–449, Feb. 2018, doi: 10.1109/JIOT.2017.2767608.

[2] J. Siegel, D. Erb, and S. Sarma, ‘Algorithms and Architectures: A Case Study in When, Where and How to Connect Vehicles’, *IEEE Intell. Transport. Syst. Mag.*, vol. 10, no. 1, pp. 74–87, 2018, doi: 10.1109/MITS.2017.2776142.

[3] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, ‘Edge computing: A survey’, *Future Generation Computer Systems*, vol. 97, pp. 219–235, Aug. 2019, doi: 10.1016/j.future.2019.02.050.

[4] A. Yousefpour *et al.*, ‘All one needs to know about fog computing and related edge computing paradigms: A complete survey’, *Journal of Systems Architecture*, vol. 98, pp. 289–330, Sep. 2019, doi: 10.1016/j.sysarc.2019.02.009.

[5] P. Varshney and Y. Simmhan, ‘Demystifying Fog Computing: Characterizing Architectures, Applications and Abstractions’, in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, Madrid, Spain, May 2017, pp. 115–124, doi: 10.1109/ICFEC.2017.20.

[6] W. Kurschl and W. Beer, ‘Combining cloud computing and wireless sensor networks’, in *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services - iiWAS ’09*, Kuala Lumpur, Malaysia, 2009, p. 512, doi: 10.1145/1806338.1806435.

[7] S. K. Dash, S. Mohapatra, and P. K. Pattnaik, ‘A Survey on Applications of Wireless Sensor Network Using Cloud Computing’, *Emerging Technologies*, vol. 1, no. 4, p. 7, 2010.

[8] V. Rajesh, J. M. Gnanasekar, R. S. Ponmagal, and P. Anbalagan, ‘Integration of Wireless Sensor Network with Cloud’, in *2010 International Conference on Recent Trends in Information, Telecommunication and Computing*, Mar. 2010, pp. 321–323, doi: 10.1109/ITC.2010.88.

[9] C. Barbieru and F. Pop, ‘Soft Real-Time Hadoop Scheduler for Big Data Processing in Smart Cities’, in *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, Crans-Montana, Switzerland, Mar. 2016, pp. 863–870, doi: 10.1109/AINA.2016.122.

[10] J. N. Hughes, M. D. Zimmerman, C. N. Eichelberger, and A. D. Fox, ‘A survey of techniques and open-source tools for processing streams of spatio-temporal events’, in *Proceedings of the 7th ACM SIGSPATIAL International Workshop on GeoStreaming - IWGS ’16*, Burlingame, California, 2016, pp. 1–4, doi: 10.1145/3003421.3003432.

[11] R. Evans, ‘Apache Storm, a Hands on Tutorial’, in *2015 IEEE International Conference on Cloud Engineering*, Tempe, AZ, USA, Mar. 2015, pp. 2–2, doi: 10.1109/IC2E.2015.67.

[12] I. Taleb, R. Dssouli, and M. A. Serhani, ‘Big Data Pre-processing: A Quality Framework’, in *2015 IEEE International Congress on Big Data*, New York City, NY, USA, Jun. 2015, pp. 191–198, doi: 10.1109/BigDataCongress.2015.35.

[13] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, ‘Edge Computing: Vision and Challenges’, *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct. 2016, doi: 10.1109/JIOT.2016.2579198.

[14] M. Satyanarayanan, ‘The Emergence of Edge Computing’, *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017, doi: 10.1109/MC.2017.9.

[15] E. I. Gaura, J. Brusey, M. Allen, R. Wilkins, D. Goldsmith, and R. Rednic, ‘Edge Mining the Internet of Things’, *IEEE Sensors J.*, vol. 13, no. 10, pp. 3816–3825, Oct. 2013, doi: 10.1109/JSEN.2013.2266895.

[16] A. Papageorgiou, B. Cheng, and E. Kovacs, ‘Real-time data reduction at the network edge of Internet-of-Things systems’, in *2015 11th International Conference on Network and Service Management (CNSM)*, Barcelona, Spain, Nov. 2015, pp. 284–291, doi: 10.1109/CNSM.2015.7367373.

[17] ‘Qualcomm Branches Out with Connected Car Platform’, *Microwaves & RF*, Jun. 13, 2016. https://www.mwrf.com/technologies/active-components/article/21846799/qualcomm-branches-out-with-connected-car-platform (accessed May 07, 2020).

[18] ‘Hortonworks Data Platform - Automated Install with Ambari’, p. 43, 2015.

[19] B. Amento, B. Balasubramanian, R. J. Hall, K. Joshi, G. Jung, and K. H. Purdy, ‘FocusStack: Orchestrating Edge Clouds Using Location-Based Focus of Attention’, in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Washington, DC, USA, Oct. 2016, pp. 179–191, doi: 10.1109/SEC.2016.22.

[20] ‘Edgent’. https://edgent.incubator.apache.org/ (accessed May 11, 2020).

[21] A. Shoker, J. Leitao, P. V. Roy, and C. Meiklejohn, ‘LightKone: Towards General Purpose Computations on the Edge’, p. 12.

[22] M. Stolpe, ‘The Internet of Things: Opportunities and Challenges for Distributed Data Analysis’, *SIGKDD Explor. Newsl.*, vol. 18, no. 1, pp. 15–34, Aug. 2016, doi: 10.1145/2980765.2980768.

[23] S. K. Sharma and X. Wang, ‘Live Data Analytics With Collaborative Edge and Cloud Processing in Wireless IoT Networks’, *IEEE Access*, vol. 5, pp. 4621–4635, 2017, doi: 10.1109/ACCESS.2017.2682640.

[24] X. Masip-Bruin, E. Marín-Tordera, G. Tashakor, A. Jukan, and G.-J. Ren, ‘Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems’, *IEEE Wireless Commun.*, vol. 23, no. 5, pp. 120–128, Oct. 2016, doi: 10.1109/MWC.2016.7721750.

[25] M. Chiang and T. Zhang, ‘Fog and IoT: An Overview of Research Opportunities’, *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016, doi: 10.1109/JIOT.2016.2584538.

[26] Y. Nakamura, H. Suwa, Y. Arakawa, H. Yamaguchi, and K. Yasumoto, ‘Middleware for Proximity Distributed Real-Time Processing of IoT Data Flows’, in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, Nara, Japan, Jun. 2016, pp. 771–772, doi: 10.1109/ICDCS.2016.101.

[27] H. Hromic *et al.*, ‘Real time analysis of sensor data for the Internet of Things by means of clustering and event processing’, in *2015 IEEE International Conference on Communications (ICC)*, London, Jun. 2015, pp. 685–691, doi: 10.1109/ICC.2015.7248401.

[28] W. Beaton, ‘Eclipse Krikkit’, *projects.eclipse.org*, Jan. 23, 2014. https://projects.eclipse.org/projects/technology.krikkit (accessed May 07, 2020).

[29] H. P. Sajjad, K. Danniswara, A. Al-Shishtawy, and V. Vlassov, ‘SpanEdge: Towards Unifying Stream Processing over Central and Near-the-Edge Data Centers’, in *2016 IEEE/ACM Symposium on Edge Computing (SEC)*, Washington, DC, Oct. 2016, pp. 168–178, doi: 10.1109/SEC.2016.17.

[30] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, ‘Fog of Everything: Energy-Efficient Networked Computing Architectures, Research Challenges, and a Case Study’, *IEEE Access*, vol. 5, pp. 9882–9910, 2017, doi: 10.1109/ACCESS.2017.2702013.

[31] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, and A. Kitazawa, 'FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities', *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 696–707, Apr. 2018, doi: 10.1109/JIOT.2017.2747214.

[32] S. Verma, Y. Kawamoto, Z. Md. Fadlullah, H. Nishiyama, and N. Kato, 'A Survey on Network Methodologies for Real-Time Analytics of Massive IoT Data and Open Research Issues', *IEEE Commun. Surv. Tutorials*, vol. 19, no. 3, pp. 1457–1477, 2017, doi: 10.1109/COMST.2017.2694469.

[33] V. Beltran, A. F. Skarmeta, and P. M. Ruiz, 'An ARM-Compliant Architecture for User Privacy in Smart Cities: SMARTIE—Quality by Design in the IoT', *Wireless Communications and Mobile Computing*, vol. 2017, pp. 1–13, 2017, doi: 10.1155/2017/3859836.

[34] J.-M. Bohli, A. Skarmeta, M. Victoria Moreno, D. Garcia, and P. Langendorfer, 'SMARTIE project: Secure IoT data management for smart cities', in *2015 International Conference on Recent Advances in Internet of Things (RIoT)*, Singapore, Singapore, Apr. 2015, pp. 1–6, doi: 10.1109/RIOT.2015.7104906.

[35] W. Zhang, Z. Zhang, and H.-C. Chao, 'Cooperative Fog Computing for Dealing with Big Data in the Internet of Vehicles: Architecture and Hierarchical Resource Management', *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 60–67, Dec. 2017, doi: 10.1109/MCOM.2017.1700208.

[36] L. Sanchez *et al.*, 'SmartSantander: IoT experimentation over a smart city testbed', *Computer Networks*, vol. 61, pp. 217–238, 2014.

[37] F. Bonomi, 'The Smart and Connected Vehicle and the Internet of Things', *Enabling Technologies*, p. 53, 2011.

[38] T. T. de Almeida, J. A. M. Nacif, F. P. Bhering, and J. G. R. Junior, 'DOCTraMS: A Decentralized and Offline Community-Based Traffic Monitoring System', *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2018, doi: 10.1109/TITS.2018.2839744.

[39] C. Chen, T. H. Luan, X. Guan, N. Lu, and Y. Liu, 'Connected vehicular transportation: Data analytics and traffic-dependent networking', *ieee vehicular technology magazine*, vol. 12, no. 3, pp. 42–54, 2017.

[40] 'Cisco Kinetic IoT Platform - Cisco IoT Solutions', *Cisco*. https://www.cisco.com/c/en/us/solutions/internet-of-things/iot-kinetic.html (accessed May 08, 2020).

[41] K. Sharma, B. Butler, B. Jennings, J. Kennedy, and R. Loomba, 'Optimizing the Placement of Data Collection Services on Vehicle Clusters', in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, Sep. 2018, pp. 1800–1806, doi: 10.1109/PIMRC.2018.8581027.

[42] N. P. Singh Rathore, A. Yadav, and A. Chauhan, 'Multi-Agent based Stable Clustering and Collision Detection In VANET', in *2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS)*, May 2020, pp. 1147–1150, doi: 10.1109/ICICCS48265.2020.9120900.

[43] *WORLD ECONOMIC SITUATION AND PROSPECTS*. NEW YORK: UNITED NATIONS PUBLICATIO, 2018.

[44] J. E. Siegel, D. C. Erb, and S. E. Sarma, 'A Survey of the Connected Vehicle Landscape—Architectures, Enabling Technologies, Applications, and Development Areas', *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 8, pp. 2391–2406, Aug. 2018, doi: 10.1109/TITS.2017.2749459.

[45] R. Du, C. Chen, B. Yang, N. Lu, X. Guan, and X. Shen, 'Effective Urban Traffic Monitoring by Vehicular Sensor Networks', *IEEE Transactions on Vehicular Technology*, vol. 64, no. 1, pp. 273–286, Jan. 2015, doi: 10.1109/TVT.2014.2321010.

[46] G. Dimitrakopoulos and P. Demestichas, 'Intelligent Transportation Systems', *IEEE Vehicular Technology Magazine*, vol. 5, no. 1, pp. 77–84, Mar. 2010, doi: 10.1109/MVT.2009.935537.

[47] L. Zhu, F. R. Yu, Y. Wang, B. Ning, and T. Tang, 'Big Data Analytics in Intelligent Transportation Systems: A Survey', *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–16, 2018, doi: 10.1109/TITS.2018.2815678.

[48] A. Pressas, Z. Sheng, P. Fussey, and D. Lund, 'Connected Vehicles in Smart Cities: Interworking from Inside Vehicles to Outside', in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, Jun. 2016, pp. 1–3, doi: 10.1109/SAHCN.2016.7732976.

[49] S. Djahel, R. Doolan, G. Muntean, and J. Murphy, 'A Communications-Oriented Perspective on Traffic Management Systems for Smart Cities: Challenges and Innovative Approaches', *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 125–151, Firstquarter 2015, doi: 10.1109/COMST.2014.2339817.

[50] E. Uhlemann, 'Introducing connected vehicles [connected vehicles]', *IEEE Vehicular Technology Magazine*, vol. 10, no. 1, pp. 23–31, 2015.

[51] G. De Nunzio, C. C. De Wit, P. Moulin, and D. Di Domenico, 'Eco-driving in urban traffic networks using traffic signals information', *International Journal of Robust and Nonlinear Control*, vol. 26, no. 6, pp. 1307–1324, 2016.

[52] M. Zhou, H. Jin, and W. Wang, 'A review of vehicle fuel consumption models to evaluate eco-driving and eco-routing', *Transportation Research Part D: Transport and Environment*, vol. 49, pp. 203–218, Dec. 2016, doi: 10.1016/j.trd.2016.09.008.

[53] H. J. Walnum and M. Simonsen, 'Does driving behavior matter? An analysis of fuel consumption data from heavy-duty trucks', *Transportation Research Part D: Transport and Environment*, vol. 36, pp. 107–120, May 2015, doi: 10.1016/j.trd.2015.02.016.

[54] S. L. Jamson, D. L. Hibberd, and A. H. Jamson, 'Drivers' ability to learn eco-driving skills; effects on fuel efficient and safe driving behaviour', *Transportation Research Part C: Emerging Technologies*, vol. 58, pp. 657–668, Sep. 2015, doi: 10.1016/j.trc.2015.02.004.

[55] M. Rutty, L. Matthews, J. Andrey, and T. D. Matto, 'Eco-driver training within the City of Calgary's municipal fleet: Monitoring the impact', *Transportation Research Part D: Transport and Environment*, vol. 24, pp. 44–51, Oct. 2013, doi: 10.1016/j.trd.2013.05.006.

[56] P. Barla, M. Gilbert-Gonthier, M. A. Lopez Castro, and L. Miranda-Moreno, 'Eco-driving training and fuel consumption: Impact, heterogeneity and sustainability', *Energy Economics*, vol. 62, pp. 187–194, Feb. 2017, doi: 10.1016/j.eneco.2016.12.018.

[57] J. N. Barkenbus, 'Eco-driving: An overlooked climate change initiative', *Energy Policy*, vol. 38, no. 2, pp. 762–769, Feb. 2010, doi: 10.1016/j.enpol.2009.10.021.

[58] A. Zavalko, 'Applying energy approach in the evaluation of eco-driving skill and eco-driving training of truck drivers', *Transportation Research Part D: Transport and Environment*, vol. 62, pp. 672–684, Jul. 2018, doi: 10.1016/j.trd.2018.01.023.

[59] I. Jeffreys, G. Graves, and M. Roth, 'Evaluation of eco-driving training for vehicle fuel use and emission reduction: A case study in Australia', *Transportation*

Research Part D: Transport and Environment, vol. 60, pp. 85–91, May 2018, doi: 10.1016/j.trd.2015.12.017.

[60] ‘AppNotesForCANBusApplication_v1.0.3_01.pdf’. Accessed: Feb. 18, 2019. [Online]. Available: http://www.falcom.de/uploads/media/AppNotesForCANBusApplication_v1.0.3_01.pdf.

[61] O. Avatefipour and H. Malik, ‘State-of-the-Art Survey on In-Vehicle Network Communication (CAN-Bus) Security and Vulnerabilities’, *arXiv preprint arXiv:1802.01725*, 2018.

[62] R. Malekian, N. R. Moloisane, L. Nair, B. T. Maharaj, and U. A. K. Chude-Okonkwo, ‘Design and Implementation of a Wireless OBD II Fleet Management System’, *IEEE Sensors Journal*, vol. 17, no. 4, pp. 1154–1164, Feb. 2017, doi: 10.1109/JSEN.2016.2631542.

[63] P. Dzhelekarski and D. Alexiev, ‘INITIALIZING COMMUNICATION TO VEHICLE OBDII SYSTEM’, p. 7, 2005.

[64] ‘OBD or CAN | Masternaut’. https://www.masternaut.com/blog/obd-or-can/ (accessed Jun. 14, 2019).

[65] ‘FMS-Standard’. http://bus-fms-standard.com/Truck/index.htm (accessed Jul. 08, 2020).

[66] K. Khosravinia, M. K. Hassan, R. Z. A. Rahman, and S. A. R. Al-Haddad, ‘Improved Latency of CAN Vehicle Data Extraction Method’, in *International Conference on Internet of Vehicles*, 2018, pp. 14–26.

[67] S. Ilarri, T. Delot, and R. Trillo-Lado, ‘A Data Management Perspective on Vehicular Networks’, *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2420–2460, 2015, doi: 10.1109/COMST.2015.2472395.

[68] R. Meng, C. Mao, and R. R. Choudhury, ‘Driving analytics: Will it be OBDs or smartphones?’, *Zendrive Whitepaper*, 2014.

[69] G. F. Türker and A. Kutlu, ‘Survey of smartphone applications based on OBD-II for intelligent transportation systems’, *Int. J. Eng. Research Appl*, vol. 6, no. 1, pp. 69–73, 2016.

[70] Z. Szalay *et al.*, ‘ICT in road vehicles — Reliable vehicle sensor information from OBD versus CAN’, in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, Jun. 2015, pp. 469–476, doi: 10.1109/MTITS.2015.7223296.

[71] D. Sik, T. Balogh, P. Ekler, and L. Lengyel, ‘Comparing OBD and CAN Sampling on the go with the SensorHUB Framework’, *Procedia Engineering*, vol. 168, pp. 39–42, 2016, doi: 10.1016/j.proeng.2016.11.133.

[72] L. Carnevale, A. Celesti, M. D. Pietro, and A. Galletta, ‘How to Conceive Future Mobility Services in Smart Cities According to the FIWARE frontierCities Experience’, *IEEE Cloud Computing*, vol. 5, no. 5, pp. 25–36, Sep. 2018, doi: 10.1109/MCC.2018.053711664.

[73] Y. Liu, H. Wang, M. Peng, J. Guan, J. Xu, and Y. Wang, ‘DeePGA: A Privacy-Preserving Data Aggregation Game in Crowdsensing via Deep Reinforcement Learning’, *IEEE Internet Things J.*, pp. 1–1, 2019, doi: 10.1109/JIOT.2019.2957400.

[74] S. Latif, S. Mahfooz, B. Jan, N. Ahmad, Y. Cao, and M. Asif, ‘A comparative study of scenario-driven multi-hop broadcast protocols for VANETs’, *Vehicular Communications*, vol. 12, pp. 88–109, Apr. 2018, doi: 10.1016/j.vehcom.2018.01.009.

[75] C. Cooper, D. Franklin, M. Ros, F. Safaei, and M. Abolhasan, ‘A Comparative Survey of VANET Clustering Techniques’, *IEEE Communications Surveys Tutorials*, vol. 19, no. 1, pp. 657–681, Firstquarter 2017, doi: 10.1109/COMST.2016.2611524.

[76] N. Lyamin, A. Vinel, M. Jonsson, and B. Bellalta, ‘Cooperative Awareness in VANETs: On ETSI EN 302 637-2 Performance’, *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 17–28, Jan. 2018, doi: 10.1109/TVT.2017.2754584.

[77] D. K. and, ‘The Internet of Vehicles Based on 5G Communications’, in *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Dec. 2016, pp. 445–448, doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.105.

[78] L. Tuyisenge, M. Ayaida, S. Tohme, and L.-E. Afilal, ‘Network Architectures in Internet of Vehicles (IoV): Review, Protocols Analysis, Challenges and Issues’, in *Internet of Vehicles. Technologies and Services Towards Smart City*, vol. 11253, A. M. J. Skulimowski, Z. Sheng, S. Khemiri-Kallel, C. Cérin, and C.-H. Hsu, Eds. Cham: Springer International Publishing, 2018, pp. 3–13.

[79] S. Chen *et al.*, ‘Vehicle-to-Everything (v2x) Services Supported by LTE-Based Systems and 5G’, *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 70–76, 2017, doi: 10.1109/MCOMSTD.2017.1700015.

[80] F. Yang, S. Wang, J. Li, Z. Liu, and Q. Sun, ‘An overview of Internet of Vehicles’, *China Communications*, vol. 11, no. 10, pp. 1–15, Oct. 2014, doi: 10.1109/CC.2014.6969789.

[81] Y. L. Morgan, ‘Notes on DSRC WAVE Standards Suite: Its Architecture, Design, and Characteristics’, *IEEE Communications Surveys Tutorials*, vol. 12, no. 4, pp. 504–518, Fourth 2010, doi: 10.1109/SURV.2010.033010.00024.

[82] F. Arena, G. Pau, and A. Severino, ‘A Review on IEEE 802.11p for Intelligent Transportation Systems’, *JSAN*, vol. 9, no. 2, p. 22, Apr. 2020, doi: 10.3390/jsan9020022.

[83] ‘1609.3-2016 - IEEE Standard for Wireless Access in Vehicular Environments (WAVE) -- Networking Services’. https://standards.ieee.org/standard/1609_3-2016.html (accessed May 12, 2020).

[84] M. Miao, Q. Zheng, K. Zheng, and Z. Zeng, ‘Implementation and Demonstration of WAVE Networking Services for Intelligent Transportation Systems’, in *Internet of Vehicles – Technologies and Services*, vol. 8662, R. C.-H. Hsu and S. Wang, Eds. Cham: Springer International Publishing, 2014, pp. 130–139.

[85] ‘Home’, *EstiNet - Simulator*. https://www.estinet.com/ns/ (accessed May 11, 2020).

[86] D. B. Rawat, D. C. Popescu, G. Yan, and S. Olariu, ‘Enhancing VANET Performance by Joint Adaptation of Transmission Power and Contention Window Size’, *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, p. 9, 2011.

[87] J. Majkowski and F. C. Palacio, ‘Dynamic TXOP configuration for Qos enhancement in IEEE 802.11e wireless LAN’, in *2006 International Conference on Software in Telecommunications and Computer Networks*, Sep. 2006, pp. 66–70, doi: 10.1109/SOFTCOM.2006.329721.

[88] R. Reinders, M. van Eenennaam, G. Karagiannis, and G. Heijenk, ‘Contention window analysis for beaconing in VANETs’, in *2011 7th International Wireless Communications and Mobile Computing Conference*, Istanbul, Turkey, Jul. 2011, pp. 1481–1487, doi: 10.1109/IWCMC.2011.5982757.

[89] C. Han, M. Dianati, R. Tafazolli, R. Kernchen, and X. Shen, 'Analytical Study of the IEEE 802.11p MAC Sublayer in Vehicular Networks', *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 873–886, Jun. 2012, doi: 10.1109/TITS.2012.2183366.

[90] M. Sepulcre, J. Gozalvez, and B. Coll-Perales, 'Why 6 Mbps is Not (Always) the Optimum Data Rate for Beaconing in Vehicular Networks', *IEEE Trans. on Mobile Comput.*, vol. 16, no. 12, pp. 3568–3579, Dec. 2017, doi: 10.1109/TMC.2017.2696533.

[91] S. Cespedes, N. Lu, and X. Shen, 'VIP-WAVE: On the Feasibility of IP Communications in 802.11p Vehicular Networks', *IEEE Trans. Intell. Transport. Syst.*, vol. 14, no. 1, pp. 82–97, Mar. 2013, doi: 10.1109/TITS.2012.2206387.

[92] O. Shagdar, M. Tsukada, M. Kakiuchi, T. Toukabri, and T. Ernst, 'Experimentation Towards IPv6 over IEEE 802.11p with ITS Station Architecture', p. 7.

[93] M. T. Barros *et al.*, 'CogITS: cognition-enabled network management for 5G V2X communication', *IET Intelligent Transport Systems*, vol. 14, no. 3, pp. 182–189, Mar. 2020, doi: 10.1049/iet-its.2019.0111.

[94] S. Malik and P. K. Sahu, 'A comparative study on routing protocols for VANETs', *Heliyon*, vol. 5, no. 8, Aug. 2019, doi: 10.1016/j.heliyon.2019.e02340.

[95] Y. Meraihi, D. Acheli, and A. Ramdane-Cherif, 'QoS performance evaluation of AODV and DSR routing protocols in city VANET scenarios', in *2017 5th International Conference on Electrical Engineering - Boumerdes (ICEE-B)*, Oct. 2017, pp. 1–6, doi: 10.1109/ICEE-B.2017.8192163.

[96] E. E. Akkari Sallum, G. dos Santos, M. Alves, and M. M. Santos, 'Performance analysis and comparison of the DSDV, AODV and OLSR routing protocols under VANETs', in *2018 16th International Conference on Intelligent Transportation Systems Telecommunications (ITST)*, Oct. 2018, pp. 1–7, doi: 10.1109/ITST.2018.8566825.

[97] W. Viriyasitavat, F. Bai, and O. K. Tonguz, 'UV-CAST: An Urban Vehicular Broadcast Protocol', p. 8.

[98] N. Wisitpongphan, O. K. Tonguz, J. S. Parikh, P. Mudalige, F. Bai, and V. Sadekar, 'Broadcast storm mitigation techniques in vehicular ad hoc networks', *IEEE Wireless Communications*, vol. 14, no. 6, pp. 84–94, Dec. 2007, doi: 10.1109/MWC.2007.4407231.

[99] S. Allal and S. Boudjit, 'Geocast Routing Protocols for VANETs: Survey and Geometry-Driven Scheme Proposal', p. 18.

[100] S. Bai, J. Oh, and J. Jung, 'Context awareness beacon scheduling scheme for congestion control in vehicle to vehicle safety communication', *Ad Hoc Networks*, vol. 11, no. 7, pp. 2049–2058, Sep. 2013, doi: 10.1016/j.adhoc.2012.02.014.

[101] J. P. Morrison, 'Flow-based programming', 2012. https://www.semanticscholar.org/paper/Flow-based-programming-Morrison/4ad394f9bc25a8e0d14f76d876618b26bf8de632 (accessed May 08, 2020).

[102] 'Comparison between Flow-Based Programming and Object-Oriented Programming'. https://www.jpaulmorrison.com/fbp/oops.shtml (accessed May 08, 2020).

[103] M. Blackstock and R. Lea, 'IoT mashups with the WoTKit', in *2012 3rd IEEE International Conference on the Internet of Things*, Wuxi, Jiangsu Province, China, Oct. 2012, pp. 159–166, doi: 10.1109/IOT.2012.6402318.

[104] 'Flow Based Programming With NoFlo'. https://www.sitepoint.com/flow-based-programming-noflo/ (accessed May 08, 2020).

[105] 'Node-RED'. https://nodered.org/ (accessed May 08, 2020).
[106] W. M. Johnston, J. R. P. Hanna, and R. J. Millar, 'Advances in dataflow programming languages', *ACM Comput. Surv.*, vol. 36, no. 1, pp. 1–34, Mar. 2004, doi: 10.1145/1013208.1013209.
[107] M. Blackstock and R. Lea, 'Toward a Distributed Data Flow Platform for the Web of Things (Distributed Node-RED)', in *Proceedings of the 5th International Workshop on Web of Things - WoT '14*, Cambridge, MA, USA, 2014, pp. 34–39, doi: 10.1145/2684432.2684439.
[108] N. P. Edwards, 'THE EFFECT OF CERTAIN MODULAR DESIGN PRINCIPLES ON TESTABILITY', p. 10.
[109] N. K. Giang, M. Blackstock, R. Lea, and V. C. M. Leung, 'Developing IoT applications in the Fog: A Distributed Dataflow approach', in *2015 5th International Conference on the Internet of Things (IOT)*, Seoul, South Korea, Oct. 2015, pp. 155–162, doi: 10.1109/IOT.2015.7356560.
[110] T. Szydlo, R. Brzoza-Woch, J. Sendorek, M. Windak, and C. Gniady, 'Flow-Based Programming for IoT Leveraging Fog Computing', in *2017 IEEE 26th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, Poznan, Poland, Jun. 2017, pp. 74–79, doi: 10.1109/WETICE.2017.17.
[111] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, 'Fog Orchestration for Internet of Things Services', *IEEE Internet Comput.*, vol. 21, no. 2, pp. 16–24, Mar. 2017, doi: 10.1109/MIC.2017.36.
[112] 'IoT Analytics Across Edge and Cloud Platforms - IEEE Internet of Things'. https://iot.ieee.org/newsletter/may-2017/iot-analytics-across-edge-and-cloud-platforms.html (accessed May 08, 2020).
[113] 'FBP inspired data flow syntax: The missing piece for the success of functional programming? | RIL Labs'. https://rillabs.com/posts/fbp-data-flow-syntax (accessed May 11, 2020).
[114] 'Apache NiFi Expression Language Guide'. https://nifi.apache.org/docs/nifi-docs/html/expression-language-guide.html#structure (accessed Dec. 30, 2016).
[115] N. Narkhede, G. Shapira, and T. Palino, 'Kafka: The Definitive Guide', p. 118.
[116] 'The World's Most Popular Data Science Platform', *Anaconda*. https://www.anaconda.com/ (accessed May 11, 2020).
[117] T. Hauck, *scikit-learn Cookbook*. 2014.
[118] 'Home - TPOT'. https://epistasislab.github.io/tpot/ (accessed May 11, 2020).
[119] R. Young, S. Fallon, and P. Jacob, 'An Architecture for Intelligent Data Processing on IoT Edge Devices', in *2017 UKSim-AMSS 19th International Conference on Computer Modelling Simulation (UKSim)*, Apr. 2017, pp. 227–232, doi: 10.1109/UKSim.2017.19.
[120] R. Young, S. Fallon, and P. Jacob, 'A Governance Architecture for Self-Adaption & Control in IoT Applications', in *2018 5th International Conference on Control, Decision and Information Technologies (CoDIT)*, Thessaloniki, Apr. 2018, pp. 241–246, doi: 10.1109/CoDIT.2018.8394824.
[121] R. Young, S. Fallon, and P. Jacob, 'Dynamic Collaboration of Centralized & Edge Processing for Coordinated Data Management in an IoT Paradigm', in *2018 IEEE 32nd International Conference on Advanced Information Networking and Applications (AINA)*, Krakow, May 2018, pp. 694–701, doi: 10.1109/AINA.2018.00105.
[122] G. Vetticaden, *georgevetticaden/hdp*. 2019.

[123] ‘Stay Alert! The Ford Challenge’. https://kaggle.com/c/stayalert (accessed May 11, 2020).

[124] R. Young, Dr. S. Fallon, Dr. P. Jacob, and Dr. D. O. Dwyer, ‘Vehicle Telematics and its Role as a Key Enabler in the Development of Smart Cities’, *IEEE Sensors J.*, pp. 1–1, 2020, doi: 10.1109/JSEN.2020.2997129.

[125] Q. Ahmed, A. I. Bhatti, and M. Iqbal, ‘Virtual sensors for automotive engine sensors fault diagnosis in second-order sliding modes’, *IEEE Sensors Journal*, vol. 11, no. 9, pp. 1832–1840, 2011.

[126] J. E. Meseguer, C. T. Calafate, J. C. Cano, and P. Manzoni, ‘Assessing the impact of driving behavior on instantaneous fuel consumption’, in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, Las Vegas, NV, USA, Jan. 2015, pp. 443–448, doi: 10.1109/CCNC.2015.7158016.

[127] J. E. Meseguer, C. K. Toh, C. T. Calafate, J. C. Cano, and P. Manzoni, ‘Drivingstyles: a mobile platform for driving styles and fuel consumption characterization’, *Journal of Communications and Networks*, vol. 19, no. 2, pp. 162–168, Apr. 2017, doi: 10.1109/JCN.2017.000025.

[128] V. Ribeiro, J. Rodrigues, and A. Aguiar, ‘Mining geographic data for fuel consumption estimation’, in *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, 2013, pp. 124–129.

[129] N. R. Abdullah, H. Ismail, Z. Michael, A. Ab. Rahim, and H. Sharudin, ‘EFFECTS OF AIR INTAKE TEMPERATURE ON THE FUEL CONSUMPTION AND EXHAUST EMISSIONS OF NATURAL ASPIRATED GASOLINE ENGINE’, *Jurnal Teknologi*, vol. 76, no. 9, Sep. 2015, doi: 10.11113/jt.v76.5639.

[130] E. Choi and E. Kim, ‘Critical aggressive acceleration values and models for fuel consumption when starting and driving a passenger car running on LPG’, *International Journal of Sustainable Transportation*, vol. 11, no. 6, pp. 395–405, Jul. 2017, doi: 10.1080/15568318.2016.1262928.

[131] E. Ericsson, ‘Independent driving pattern factors and their influence on fuel-use and exhaust emission factors’, *Transportation Research Part D: Transport and Environment*, vol. 6, no. 5, pp. 325–345, Sep. 2001, doi: 10.1016/S1361-9209(01)00003-7.

[132] R. Young, S. Fallon, P. Jacob, and D. O. Dwyer, ‘A Flow Based Architecture for Efficient Distribution of Vehicular Information in Smart Cities’, in *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, Granada, Spain, Oct. 2019, pp. 93–98, doi: 10.1109/IOTSMS48152.2019.8939233.

[133] R. Young, S. Fallon, and P. Jacob, ‘WAVE-Flow; an Architecture for Distribution and Governance of Vehicular Information in Smart Cities.pdf’. Submitted to IET Smart Cities, Jul. 2020.

[134] S. Y. Wang *et al.*, ‘NCTUns 4.0: An Integrated Simulation Platform for Vehicular Traffic, Communication, and Network Researches’, in *2007 IEEE 66th Vehicular Technology Conference*, Sep. 2007, pp. 2081–2085, doi: 10.1109/VETECF.2007.437.

[135] S.-Y. Wang and C.-C. Lin, ‘NCTUns 5.0: A Network Simulator for IEEE 802.11(p) and 1609 Wireless Vehicular Network Researches’, in *2008 IEEE 68th Vehicular Technology Conference*, Sep. 2008, pp. 1–2, doi: 10.1109/VETECF.2008.464.

[136] S. Wang and C. Lin, ‘NCTUns 6.0: A Simulator for Advanced Wireless Vehicular Network Research’, in *2010 IEEE 71st Vehicular Technology Conference*, May 2010, pp. 1–2, doi: 10.1109/VETECS.2010.5494212.

[137] *Synthesis of Freight Research in Urban Transportation Planning*. Transportation Research Board, 2013.

[138] V. Astarita, V. Giofré, D. Festa, G. Guido, and A. Vitale, 'Floating Car Data Adaptive Traffic Signals: A Description of the First Real-Time Experiment with "Connected" Vehicles', *Electronics*, vol. 9, no. 1, p. 114, Jan. 2020, doi: 10.3390/electronics9010114.

[139] Jia Lei, K. Redmill, and U. Ozguner, 'VATSIM: a simulator for vehicles and traffic', in *ITSC 2001. 2001 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.01TH8585)*, Aug. 2001, pp. 686–691, doi: 10.1109/ITSC.2001.948743.