Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

A Communication Architecture for Transportation Digital Twins using Apache Kafka

Conor Fennell
Academic Supervisor: Prof. Vinny Cahill

April 19, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Electronic & Computer Engineering)

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Abstract

A digital twin is a virtual representation of a system that is updated from real-time data. A digital twin can provide accurate information about the current state of the system, and allow for predictions of future states of the system.

Motorway traffic has been growing year on year with the ever increasing people and vehicle population. With this growth comes the risk of increased congestion, traffic accidents, and roadworks. These issues could be reduced with real-time traffic monitoring. A method of real-time traffic monitoring could be possible with a digital twin of the motorway. In order to create a digital twin, traffic data from the motorway is needed. This data must be real time and readily available. Several sensors exist on modern motorways such as inductive loops, road gantry cameras, and radar detectors. Others data sources can be found directly on vehicles such as GPS devices. The data from these sensors can be used to feed the digital twin.

Apache Kafka is a distributed messaging service that is based on a publisher-subscriber architecture. In Apache Kafka, data can be partitioned into 'partitions' that are within different 'topics'. This research explores the use of Apache Kafka as the communication service between the motorway sensors and the digital twin. The objective is to create a highly available, high throughput, and low latency design. The proposed system implements topics per sensor group and partitioning per sensor. The research also explores the properties of the motorway sensors, focusing on their latency and the type of data they produce.

The proposed Apache Kafka data delivery architecture is evaluated using simulated data streams. These simulated data streams are taken from virtual sensors which draw data from a SUMO simulation of the M50 motorway in Dublin. Several simulations are conducted using various Kafka configurations. The effects of compression, batching, and varying the broker hardware are tested, as well as varying the number of brokers and consumers. The effects of the relative distance between the clients and Kafka are also tested.

It was found that single broker designs offer the highest performance in terms of latency, however these systems have a single point of failure. It was found that compression can decrease the latency by approximately 7%. Compression also increases the throughput of the system by 40%. The use of multiple consumers also improves the throughput of the system by 21%. Reducing the distance between Kafka and its clients reduced the latency of the system, however throughput is unchanged.

The final solution is a 3 broker configuration. This system uses compression to decrease the latency, and replication across the brokers to increase the availability of the service. The final solution has a dedicated consumer per topic. This system experiences 145 ms of latency in low traffic simulations, and 156 ms latency in high traffic simulations with ideal cluster placement.

Acknowledgements

I would like to thank Prof. Vinny Cahill for his continued support and guidance throughout the duration of the project. His bi-weekly meetings and advice helped lead me in the right direction through all stages of the project.

I would like to thank my friends and family for their relentless support over the years.

Finally, I would like to thank my girlfriend Shauna for her unwavering positivity and patience during not only this dissertation, but throughout my degree.

Contents

List of Figures

List of Tables

Nomenclature

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
ATCS	Active Traffic control System
AWS	Amazon Web Services
CAV	Connected Autonomous Vehicles
CPU	Central Processing Unit
CSV	Comma Separated Values
CV	Connected Vehicles
EC2	Elastic Compute Cloud
FOV	Field of View
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GTFS	General Transit Feed Specification
GZIP	GNU Zip
HDFS	Hadoop Distributed File System
HGV	Heavy Goods Vehicle
IQR	Interquartile Range
ITS	Intelligent Transport Systems
MITSIM	Microscopic Traffic Simulator
MQTT	MQ Telemetry Transport Protocol
OS	Operating System
PDF	Portable Document Format
PRF	Pulse Repetition Frequency
Pub-Sub	Publisher-Subscriber system
PVT	Position, Velocity, Time
RAM	Random Access Memory
RFID	Radio Frequency Identification
RSU	Roadside Unit
S3	Simple Storage Service
SNAP	Type of Network Attached Storage
SSL	Secure Sockets Layer
STOMP	Simple Text Orientated Messaging Protocol
SSS	Speed Safety System
SUMO	Simulation of Urban Mobility

TCP	Transmission Control Protocol
TII	Transport Infrastructure Ireland
Tp	Throughput
TraCI	Traffic Control Interface
UTF-8	Unicode Transformation Format – 8-bit
XML	Extensible Markup Language

1 Introduction

This dissertation presents a design for a communication architecture between motorway traffic sensors and a digital twin. The requirements of this system is that the data is highly available and is streamed with low latency and high throughput. The background research conducted as part of this dissertation justify the technology choices made throughout the design. These choices include that of the virtual sensor design and that of the choice of middleware Apache Kafka.

1.1 Motivation

A digital twin is a virtual representation of a system that is updated from real-time data. This can provide accurate information about the current state of the system and allows for predictions of future states of the system. Digital twin's can then be used to feed machine learning models, or aid in decision making [IBM Digital Twin Exchange, 2021].

Irish roads are getting busier and busier each year [Central Statistics Office, 2019]. This leads to congestion. This is particularly evident on Dublin's M50 motorway, which has over 400,000 unique journeys everyday [Carswell, 2017].

A digital twin of a motorway could be used to predict the traffic volume ahead of time and aid traffic management authorities in reducing congestion on the motorway. Other benefits of a motorway digital twin include incident prediction and predicting the best times for roadworks and lane closures.

In order to create a digital twin of a motorway there will need to be real-time data feeds of traffic information that can be used in the digital twin. This data needs to be communicated to the digital twin with low latency. This data may be gathered from various sensors that cover the motorway. These sensors may include inductive loop traffic counters, roadside cameras, GPS data, toll-bridge data, as well as other data sources.

1.2 Objectives

The goal of this project is to design a framework that can take real-time sensor data from a motorway and use that data in order to feed a digital twin. This framework will essentially be the middleware between the physical sensors on a given motorway, and the digital twin model of that motorway.

1.3 Approach

The middleware system is implemented using a publisher-subscriber architecture (Pub-Sub). This architecture is a *'style of messaging application in which the providers of information (publishers) have no direct link to specific consumers of that information (subscribers), but the interactions between publishers and subscribers are controlled by pub/sub brokers'* [IBM, 2021]. Apache Kafka is the chosen middleware technology for this project.

The communication architecture is tested using simulated data streams which are representative of practical data sources available on the M50 motorway. A SUMO simulation of the M50 is used as the common source of traffic data to evaluate the communication framework. The simulation is a 4-lane 7km segment of the M50 which includes 2 major junctions.

Possible sources of real time data relating to motorways are identified. These data sources may include inductive loop traffic counters, toll-bridge data, GPS data, and traffic camera feeds. Virtual sensors are created. These virtual sensors are designed to be representative of sensors which may be found on a motorway. These sensors will produce simulated data streams which are representative of these data sources. These sources should model the quality and type of data from a specific source, as well as the expected latency that would be observed from the real data stream.

This data will be transferred from the virtual sensors to the digital twin using the designed framework. This digital twin will be representative of the M50 motorway, however the design of the framework is applicable to any motorway scenario.

The communication framework is tested using several simulation scenarios with varying Kafka configurations. The performance of the system is assessed in terms of its throughput, latency, and availability.

1.4 Challenges

The challenges faced by researchers attempting to build a message delivery architecture for transportation digital twins are:

- Selecting the appropriate message delivery service.
- Understanding the properties of various sensors which exist on the road network. The data format and associated latency and processing time of the sensors is extremely important for this project in order to create the virtual sensors.
- Evaluate the performance of the design using the SUMO M50 simulation and the virtual sensors which create the data streams for the digital twin.

1.5 Other Considerations

There are a number of variables which are worth noting but are not considered in the design of this project:

- Privacy concerns associated with third party data, mainly probe vehicle data which may be provided by companies such as Google or TomTom.
- Concerns with security due to transferring a large number of messages over the network. Apache Kafka is capable of utilising security protocols such as SSL, however this is not explored in this project.

1.6 Report Outline

The remainder of this dissertation is structured as follows:

- **Chapter 2 - Background:** This section contains research on available technologies relevant to the project and justification for the technologies adopted in the final design.
- **Chapter 3 - Methodology & Implementation:** Outlines the design of the implementation.
- **Chapter 4 - Evaluation:** Evaluates the performance of the previously defined implementation.
- **Chapter 5 - Summary:** Summarises the dissertation.
- **Chapter 6 - Conclusion:** Re-iterates the key findings and results, and discusses future work in this area.

2 Background

This chapter presents background information that is relevant to the project. It contains a brief overview of digital twins, and comparisons of current middleware technologies which may be useful for this application. It also contains research on motorway sensors which can be used to feed the digital twin, as well as an overview of SUMO. It also contains a literature review of current work in the field of digital twins and intelligent transport systems (ITS).

2.1 What is a Digital Twin?

A digital twin is a virtual real-time model that accurately represents a system. A digital twin proves accurate information about the current state of the system. This may allow for predictions of future states of the system. Digital twins can vary in their application and design, however digital twins usually consist of 3 distinct parts. There is the physical system in which the digital twin is trying to represent, the virtualised product of this system, and the communication from the physical to the virtual system. This project focuses on the communication from the physical to the virtual system. The following sections outlines suitable middleware for this application.

2.2 Middleware

There are a number of middleware technologies that could potentially be used for this project. Traditional distributed file systems such as such as HDFS [Hadoop, 2008] and cloud storage systems such as Amazon AWS S3 [Amazon Web Services, 2022a] are not capable of meeting the real-time processing requirements needed for this project [Guo et al., 2021]. For this project, publish subscribe systems are of great interest as they can receive and send data to and from various sources in parallel while maintaining low latency and high throughput

2.2.1 An Overview of Publish-Subscriber systems

Publish subscribe (pub-sub) systems usually consist of 3 key components:

- **Publishers (Producers):** Producers publish messages to the broker. Messages are published to specific groups called 'topics' that exist on the broker. In this application the motorway sensors act as the producers.
- **Subscribers (Consumers):** Consumers subscribe to message streams from the broker. The specific messages they receives are determined by the 'topics' to which they are subscribed. In this project the digital twin is the consumer.
- **Broker:** A broker is the server that facilitates the messaging service. Brokers manage the data among the different topics. Brokers interact with the producers and consumers in order to deliver the messages from producer to consumer. A group of brokers is referred to as a cluster.

A key feature of pub-sub systems is that there is entity decoupling. This means that producers and consumers are not aware of each other. There is also time decoupling between producers and consumers. They do not need to be producing and consuming at the same time. The interaction between either producer or consumer and the pub-sub infrastructure does not synchronously block the producer or consumer execution threads, allowing maximum usage of processor resources at producers and consumers alike. This is known as synchronisation decoupling [Dobbelaere et al., 2017].

2.2.2 What is Apache Kafka?

Apache Kafka is a popular and highly scalable publisher-subscriber platform. Kafka's design allows for high throughput and low latency which makes it an ideal candidate for this project. In Kafka, topics can be divided into partitions, with the messages in a partition guaranteed to be ordered. Messages from different partitions or topics are not guaranteed to be ordered. Multiple consumers can simultaneously read messages from one or more partitions to improve parallel processing capabilities [Dobbelaere et al., 2017]. Sample code taken from [Kreps et al., 2011] of the producer is given below. A message is defined to contain just a payload of bytes. Messages can be encoded using several serialization methods. Producers can batch sets of messages in a single publish request for improved efficiency.

```
producer = new Producer(...);
message = new Message('test message str'.getBytes());
set = new MessageSet(message);
producer.send('topic1',set);
```

Figure 2.1: Sample producer code [Kreps et al., 2011]

Sample consumer code which is also taken from [Kreps et al., 2011] outlines how a consumer would consume messages from the topic labelled 'topic1' which was produced using the sample producer code.

```
streams[] = Consumer.createMessageStreams('topic1', 1)
for (message : streams[0]) {
bytes = message.payload();
// do something with the bytes
}
```

Figure 2.2: Sample consumer code [Kreps et al., 2011]

Since Kafka is distributed by design, there is usually more than one broker. Groups of multiple brokers are known as Kafka clusters. Using a cluster there is inherent load balancing as a topic is divided into multiple partitions and each broker stores one or more of those partitions. Multiple producers and consumers can publish and retrieve messages at the same time [Kreps et al., 2011]. Figure 2.1 which is originally presented in [Dobbelaere et al., 2017] displays the architecture of Apache Kafka.

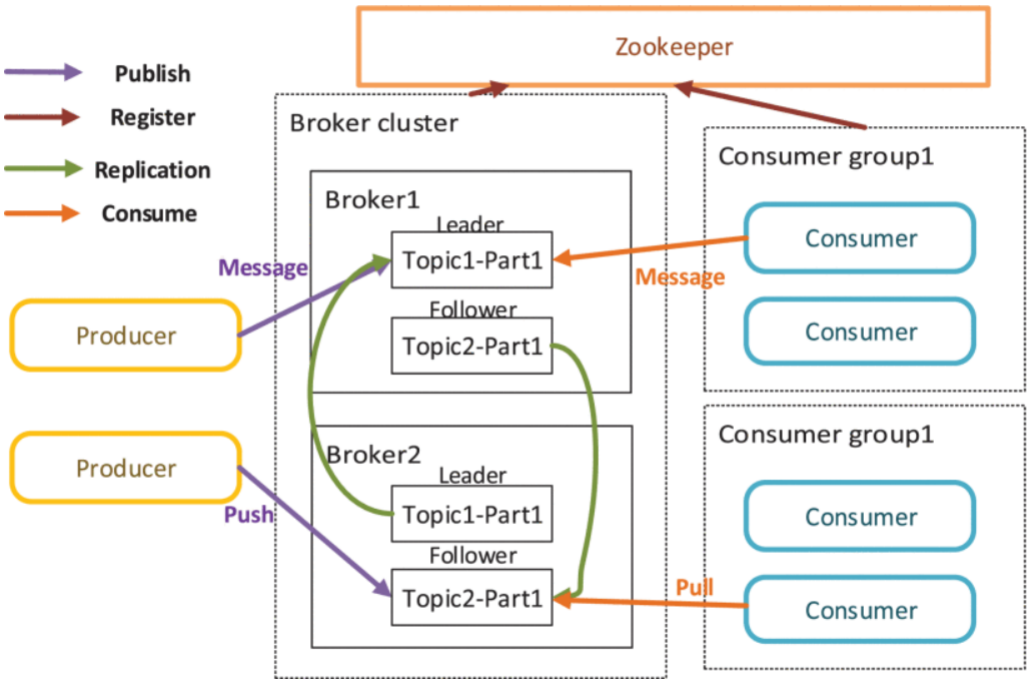


Figure 2.3: Kafka architecture [Dobbelaere et al., 2017]

The most important requirement's of Kafka for this project are that it can deliver a large number of messages from the sensors while maintaining low latency. There are a number of other services that can deliver messages with low latency, however Kafka stands out as it can continue to deliver messages with low latency while maintaining high throughput.

Kafka's high throughput can be attributed to three factors.

1. Kafka makes use of zero-copy operations [Kreps et al., 2011]. This means that Kafka writes data directly from the read buffer to the socket buffer. Traditional systems require data to be copied to and from the application before being written to the socket buffer. This is outlined in more detail in [Garg, 2019].
2. Kafka has the ability to batch data in order to improve the throughput [Kreps et al., 2011].
3. Kafka employs data compression techniques to improve the efficiency of data transmission.

Another important aspect of this project is the availability of the service. It is desirable to have a system which can still deliver messages despite a broker failure. Kafka makes use of replication. This means messages are copied to multiple brokers. In the event of a broker

failure, data loss is minimised as the messages are replicated across brokers. The other brokers in the cluster will continue to make these messages available to consumers. This allows for a highly available service.

ZooKeeper

Besides the main producer, consumer and broker components mentioned previously, Kafka also relies on ZooKeeper. ZooKeeper is used to track the status of Kafka and maintain a list of Kafka topics and messages [Kreps et al., 2011]. Kafka's dependence on Zookeeper was originally used to manage the locations of Kafka's brokers. As Kafka has evolved the dependency on ZooKeeper has lessened. Developers at Kafka have made it clear that ZooKeeper will eventually be deprecated. From v2.8.0 on, Kafka can be run without ZooKeeper for basic use, however the transition is not yet complete and ZooKeeper is still prevalent in most current Kafka implementations [McCabe, 2020].

2.2.3 Other middleware systems

There are several other middleware technologies that were considered for this project.

RabbitMQ

RabbitMQ is an open source implementation of the AMQP protocol, and unlike Kafka, does not rely on a third-party cluster management system like ZooKeeper. RabbitMQ has poor scalability due to the complete replication design [Guo et al., 2021].

ActiveMQ

ActiveMQ is a Java-based messaging broker with support for several communication protocols, such as MQTT, AMQP, STOMP, and WebSocket. It manages and allocates resources very efficiently and can achieve high throughput [Donta et al., 2021]. ActiveMQ does not have a lot of community support, with maintenance records and user feedback in decline [Guo et al., 2021].

Pulsar

Pulsar is an open-source distributed messaging system. It is developed in Java and based on the TCP protocol [Guo et al., 2021]. In Pulsar, the messages are stored in Bookies, that are managed by Bookkeeper [Junqueira et al., 2013]. Since Pulsar brokers do not store messages locally, it needs to connect to a BookKeeper cluster in order to search messages. This increases network overhead and has an impact on performance [Guo et al., 2021].

RocketMQ

RocketMQ is a Java-based message delivery service [Yongguo et al., 2019]. RocketMQ has the concept of a queue in each topic, which is similar to the partition concept in Kafka. The messages in the queue are guaranteed to be ordered. RocketMQ is extremely low latency, however it does not offer high throughput like Kafka. The RocketMQ community is also not very active compared to Kafka and RabbitMQ [Guo et al., 2021].

2.2.4 Comparing middleware systems

The most important metrics to consider for this project are throughput and latency. The throughput is important as there will be large quantities of data from the sensors, especially during rush hour when the number of vehicles will increase significantly. The latency of the system is also very important as the digital twin requires the data to be transferred in real time. Any increase in latency directly affects the accuracy of the digital twin. It is crucial that low latency can be delivered while maintaining high throughput.

Comparing the aforementioned services, it is clear that Kafka has the superior throughput abilities. This is backed up by producer and consumer tests conducted in [Kreps et al., 2011] that show the throughput performance of Kafka, RabbitMQ, and ActiveMQ.

They found that for for each system running a producer to publish a total of 10 million messages, each of 200 bytes, Kafka outperformed other systems with a batch size of 1 and a batch size of 50. ActiveMQ and RabbitMQ don't seem to have an easy way to batch messages and it is assumed that a batch size of 1 is used. Figure 2.2 which is originally presented in [Kreps et al., 2011] shows the throughput performance of Kafka, RabbitMQ, and ActiveMQ:

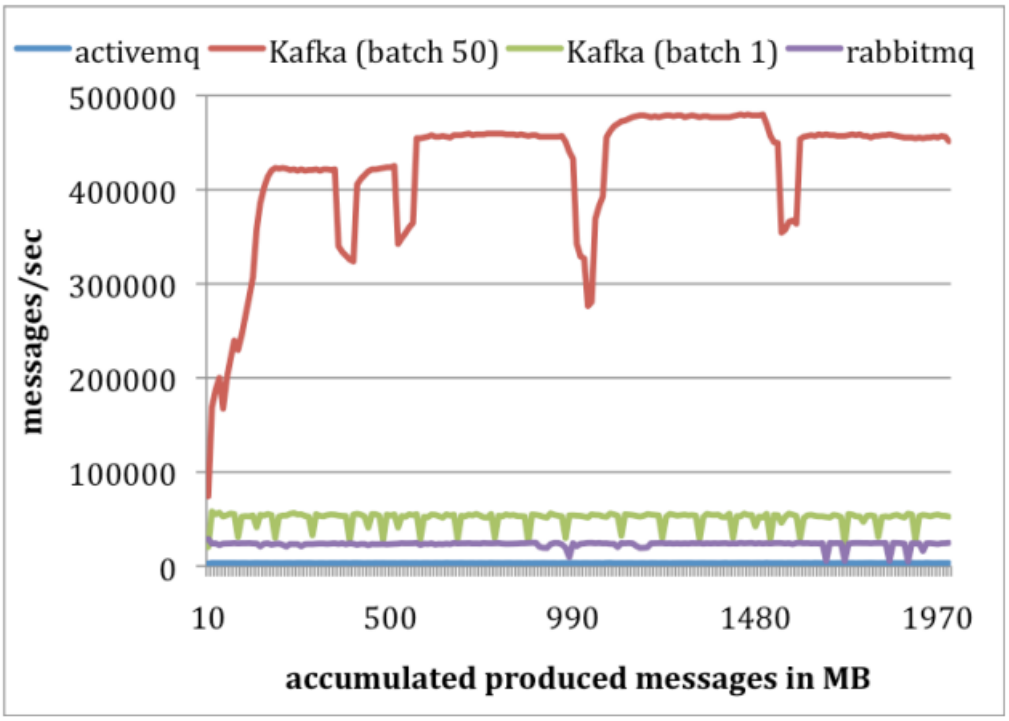


Figure 2.4: Comparing producer throughput performance [Kreps et al., 2011]

It is clear that Kafka has the best throughput performance of these systems, with the batch size of 50 the highest perform. Kafka with a batch size of 1 also significantly outperformed RabbitMQ and ActiveMQ. Kafka published messages at a rate of 50,000 per second for batch size of 1, and 400,000 messages per second for batch size of 50. This is significantly higher than RabbitMQ which managed around 25,000 messages per second, and ActiveMQ managing under 10,000 messages per second.

Consumer tests are also conducted. Similar to the producer test, all systems used a single consumer to retrieve a total of 10 million messages. Figure 2.3 presents these results which are originally found in [Kreps et al., 2011]:

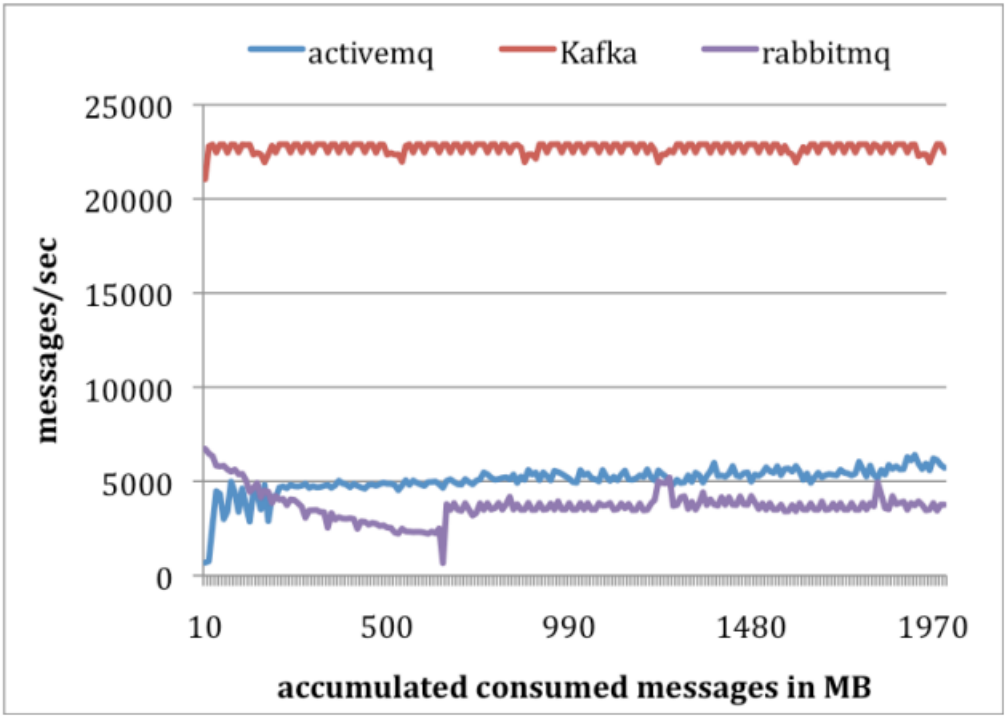
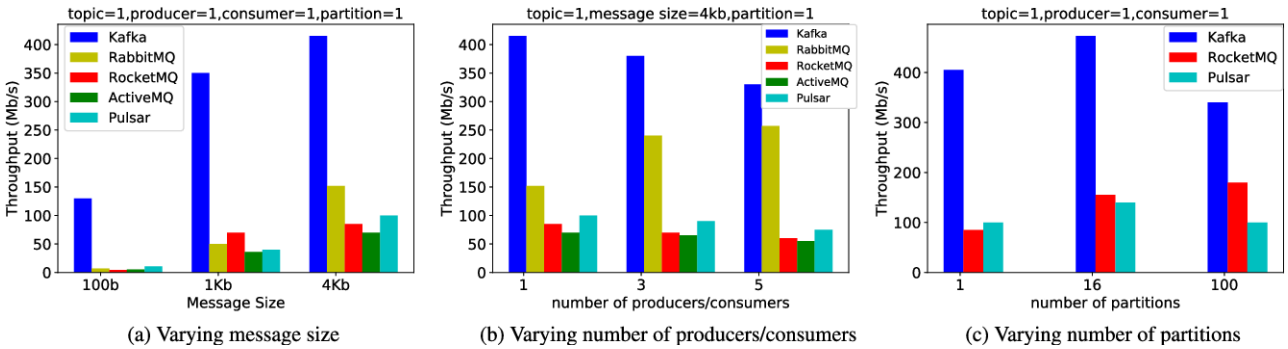


Figure 2.5: Comparing consumer throughput performance [Kreps et al., 2011]

It can be seen that Kafka consumed around 22,000 messages per second on average. This is more than 4 times that of ActiveMQ and RabbitMQ.

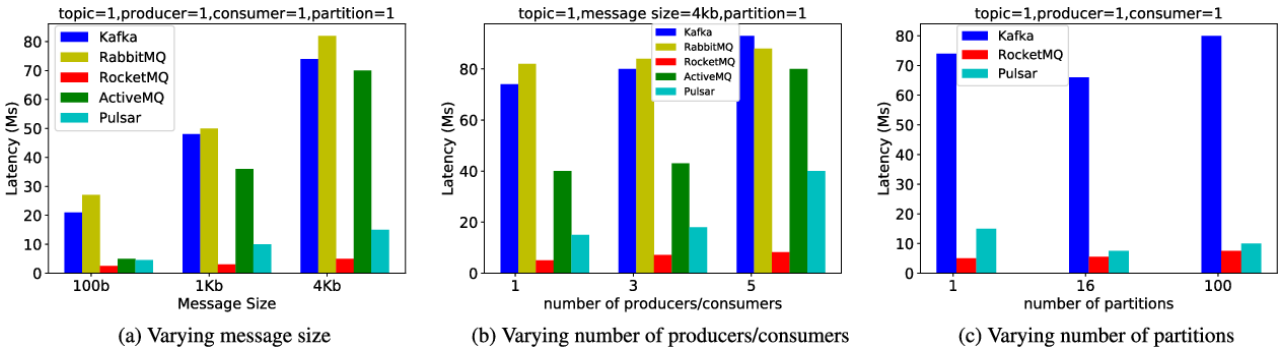
Similar tests are conducted in [Guo et al., 2021]. The throughput is recorded with varying message size and fixing the the number of topics/producers/consumers/partitions. These tests are conduced using Kafka, RabbitMQ, ActiveMQ, Pulsar, and RocketMQ. It is clear from Figure 2.6 that Kafka displays the best throughput performance across all tests [Guo et al., 2021].



(a) Varying message size (b) Varying number of producers/consumers (c) Varying number of partitions

Figure 2.6: Comparing throughput with varying configurations [Guo et al., 2021]

Latency tests are also conducted in [Guo et al., 2021]. These tests show the effect of the number of producers/consumers on the latency result. The number of partitions and the number of topics are fixed at 1, the message size is fixed as 4KB, and the number of producers/consumers is varied from 1 to 5. It can be seen that Kafka, RabbitMQ, and ActiveMQ show longer latency than the other systems. Figure 2.7 originally in [Guo et al., 2021] can be seen below:



(a) Varying message size (b) Varying number of producers/consumers (c) Varying number of partitions

Figure 2.7: Comparing latency with varying configurations [Guo et al., 2021]

It may appear that Kafka does not fulfil the low latency requirement for this project, however it is important to note that the tests conducted in Figure 2.7 do not take throughput into account. Researchers [Alok Nikhil, 2020] found that Kafka has lower latency at higher throughput than RabbitMQ, and that Kafka has lower latency than Pulsar in all cases. It also found that RabbitMQ can achieve lower end-to-end latency than Kafka, but only at significantly lower throughput. This can be seen in Figure 2.8 [Alok Nikhil, 2020]:

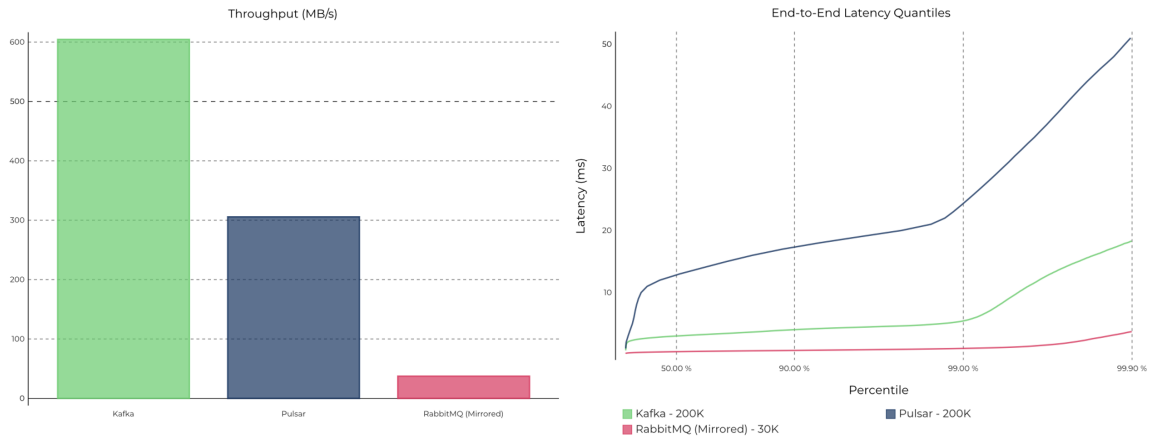


Figure 2.8: Throughput & latency comparisons between Kafka, RabbitMQ, & Pulsar [Alok Nikhil, 2020]

There are a number of message delivery guarantees that need to be defined in order to compare the middleware systems.

- *At-most-once:* Messages will be delivered at most once. There is no guarantee the message will be delivered, and once it is delivered there is no chance of delivering the message again.
- *At-least-once:* This guarantee's message delivery, however there may be duplicate messages.
- *Exactly-once:* As the same suggests, this means a message will be delivered exactly one time. This is difficult to achieve in practise.

The following table is compiled from information found in [Kreps et al., 2011] [Guo et al., 2021] [Alok Nikhil, 2020]. The table outlines the properties of each of the middleware systems.

	Kafka	**RabbitMQ**	**RocketMQ**	**ActiveMQ**	**Pulsar**
Develop Language	Scala & Java	Erlang	Java	Java	Java
Comm. Protocol	TCP	TCP	Custom	Multiple	TCP
Delivery guarantee	ALL*	at-most-once	at-least-once	at-least-once	ALL*
Order guarantee	Partition-order	None	Queue-order	Queue-order	Global-order
Throughput	Excellent	Good	Poor	Poor	Poor
Latency (low tp)	Good	Good	Excellent	Good	Excellent
Latency (high tp)	Good	Poor	Good	Poor	Poor

Table 2.1: Middleware system comparisons

*at-most-once/at-least-once/exactly-once

2.2.5 Evaluating choice of middleware

Apache Kafka is selected as the technology of choice for this project. This is for several reasons.

1. Kafka displays the best throughput of all the aforementioned middleware [Kreps et al., 2011] [Guo et al., 2021] [Alok Nikhil, 2020].
2. Kafka is low latency. There are other systems that offer similar or better latency performance such as RabbitMQ and RocketMQ, however it has been shown for high throughput applications Kafka outperforms these systems [Kreps et al., 2011] [Guo et al., 2021] [Alok Nikhil, 2020].

3. Kafka partitions are order guaranteed. Some systems such as RabbitMQ do not guarantee order. This is important for the digital twin application as messages being delivered in the wrong order increases the complexity of implementation [Guo et al., 2021].
4. Partitions in Kafka are an excellent tool for sorting data as well as load-balancing among brokers. The number of partitions in a topic in Kafka can be accurately set, then randomly and evenly distributed on the cluster, and the number of replicas can be freely defined. Many other systems have message queues such as RabbitMQ, but they are not as configurable as Kafka.
5. Kafka has excellent community support and documentation. The same can not be said for other systems such as RocketMQ and Pulsar. ActiveMQ has also seen decline in its user community, as user feedback and maintenance records have declined over the years [Guo et al., 2021].

2.3 Motorway Sensors

The aim of this section is to specify the types of available traffic sensors on today's roads and identify the potential data streams these sensors produce which could be used to feed a digital twin model. The design of each sensor is discussed as well as the quality of data and type of data streams these devices produce.

2.3.1 GPS/GNSS

GPS satellites circle the earth two times a day in a particular orbit. Each satellite transmits a unique signal and orbit parameters that allows a GPS device to decode and compute the precise location of the satellite. GPS receivers use this information and trilateration to calculate a user's exact location. The GPS device measures the distance to each satellite by measuring the time taken to receive a transmitted signal. With measurements from multiple satellites, the device can determine the user's location [Garmin, 2021a].

GPS enabled smartphones are typically accurate up to 4.9 metres [US Government, 2021]. The latency for a GPS measurement is typically in the range of milliseconds between for the GPS device and the satellites. The PVT (position, velocity, time) output rate of a typical GNSS receiver is in the range of 1HZ to 10HZ [NVS, 2021], that is a device can output a unique data point between 1 and 10 times per second. GPS does not provide any data on the type or class of vehicle.

There are a number of services that utilise GPS technology to provide users with location data. The most common of these are Sat Nav systems made by companies such as Garmin and TomTom, and mobile applications such as Google maps.

Sat Nav Systems

Sat Navs can provide the user with their current location, routes, journey time estimation etc. TomTom claims their road data is collected every 1 second, and is made available to their customers every 1 minute or less [TomTom, 2012]. This data that is made available does not contain information about individual vehicles, but rather about the flow and volume of traffic, however it may be possible to get this data from TomTom. Garmin also provides similar traffic flow data that they collect from their devices and make available to users every 60 seconds. Once again this data is processed and modelled in their data centres before being supplied to the devices, however it also may be possible to obtain the unprocessed data [Garmin, 2021b].

Assuming strong wireless connection, and the central server and digital twin host both being in a 50km radius, an estimated 500ms latency time may be observed between the vehicle and the digital twin with a sampling frequency of 1HZ using a Garmin 16x GPS sensor as a baseline for our estimations [Garmin, 2008].

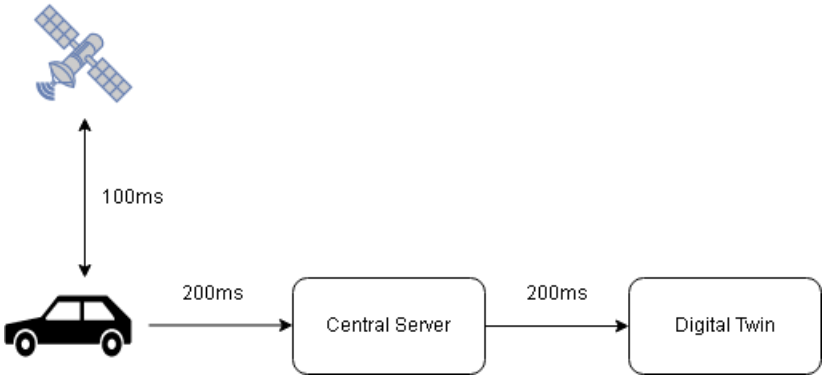


Figure 2.9: Latency estimation for GPS connected vehicle to digital twin Garmin [2008]

Google Maps

Google Maps allows for a full stream of the vehicle's journey in real time. Google maps uses a dedicated GPS receiver built into most modern mobile phones, as well as Wi-Fi modems and cellular towers to track the device's location [Google, 2021a]. Google Maps is accurate to around 20 metres according to Google [Google, 2021b]. Google maps provides the user with location data that is updated at 1HZ. This data is processed and sent to Google's remote servers about once a minute, however it is possible to increase this frequency to 15 seconds in Google maps 'Start driving mode' setting [Google, 2021c].

Probe Vehicles

A probe vehicle is a vehicle that can be used as a moving traffic sensor [Linnartz, 2010]. Probe vehicles may be used in conjunction with other data streams to provide insight into the flow of traffic. Any vehicles with live and available data feeds can be used as probe vehicles. These may include public buses, taxis, delivery vans, and commercial vehicles with tracking etc.

Public busses in Ireland are tracked by GPS using the GTFS (General transit feed specification) protocol [Government of Ireland, 2020]. This provides real-time bus information for passengers such as bus estimated arrival time, at which stop the bus is currently etc. This real-time information is available on a number of different mobile applications. However the API for accessing the data is available to developers.

The API can provide the location of a bus and it's route. While this technology only provides information on the busses, this data can still be very powerful as the bus information can be used to make estimations of the flow of traffic by treating the buses as probe vehicles. For example if the bus is moving very slowly in a particular area where there is no bus stop, this may indicate that there is heavy traffic in this area. This data would best be used as supplementary data to be used in combination with other data feeds. The API collects data from buses at a frequency of 1HZ. This is PVT data with the intended route and arrival times of the bus also being included. The location is accurate to around 20 metres.

DHL also provides live tracking for customers. This allows customers to track their parcels live in the DHL app [DHL Paket GmbH, 2012]. It is unclear what system is used to provide this service, however it could be assumed that it provides similar performance to Google maps services, with a 1HZ refresh time for customers and accuracy up to 20 metres. It may be possible to obtain the PVT data from the live tracking from DHL to use their delivery vehicles as probe vehicles.

2.3.2 Radar

Radio waves are emitted from the sensor. Waves can be deflected by target objects. Some of these waves are reflected back to the sensor. Signal processing tools can be used to determine if an object has been detected and it can reveal some characteristics about the target object.

On Vehicle Radar

Radar can provide different data depending on where the sensor is located. Some vehicles use radar as part of their self-driving systems. The sensors are used for object detection around the vehicle. This allows the vehicle to sense other vehicles in the area. It can also get the relative velocity of the other vehicles, however this alone does not provide the location of the vehicle without another sensor (such as GPS).

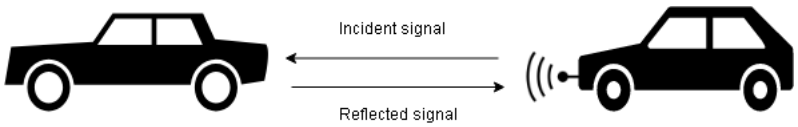


Figure 2.10: On vehicle radar used to detect other vehicles

Roadside Radar

Another method of using radar for this problem would be to mount radar sensors to the side or overhead on the motorway. This would allow the sensors to detect vehicles speed, lanes, and direction of travel, as well as providing their location since the sensors will be in known fixed locations.

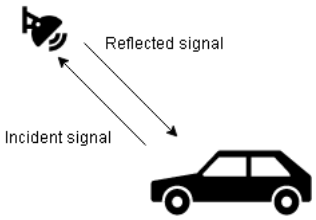


Figure 2.11: Roadside radar detector operations

Vehicle radar is accurate up to around 1 kilometer, however most devices have a much smaller usable range than this. The output rate of radar detectors is called the pulse repetition frequency (PRF) [Cambridge Pixel, 2019]. Taking a popular roadside radar device as an example, the TSR10, that is used for single lane vehicle detection. It is accurate to 30 meters, and can measure the speed and distance of an oncoming vehicle. It has a refresh rate of 20HZ. This means that the device can potentially make 20 measurements per second [Nanoradar, 2020]. The location of the device would also be known since it is a fixed device. This is the data that could be accessed if there was a direct feed from the radar device.

However in the case of there being middleware in between the feed and the device, the data may differ. It would be more common to receive data from a central source where data from multiple sources is gathered. This would add considerably more delay to the stream of data.

Radar devices are non-intrusive to install. This means that lanes would not need to be closed to install and perform maintenance of the devices. This is a significant advantage compared to some other sensor types (such as inductive loops).

Electroautomation's Traffic SSS (Speed Safety System) uses multi tracking radar technology to monitor a road stretch of 150 metres over several lanes. It is capable of classifying vehicles based on the frequency response and can return high resolution digital images [Electro Automation, 2021a]. These devices are connected to SNAP software that is used for data analytics for traffic management. This software can provide real-time traffic reports from the sensors. The server software requires Gigabit Ethernet connection [Sensy Networks, 2021].

Assuming gigabit Ethernet connection between radar device and server, and strong wireless connection from server to digital twin model, with both the server and digital twin host being in a 50km radius, an estimated 210ms latency time may be observed between the vehicle and the digital twin with a sampling frequency of 21HZ.

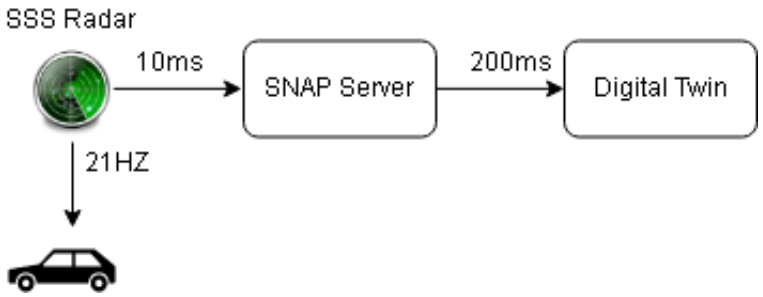


Figure 2.12: Latency estimation vehicle to digital twin using SSS radar detector [Electro Automation, 2021a]

2.3.3 Cameras

Roadside Cameras

Roadside cameras could provide rich data about the traffic flow. With video processing it is possible to extract metrics about the flow of traffic. Cameras can provide the direction a vehicle is travelling, the vehicle's speed, the lane a vehicle is in, as well as the type of vehicle and a timestamp of the data. Cameras can also read the registration of vehicles, which may not necessarily be useful for our digital twin application. The location would also be known as the camera would be mounted in a fixed location.

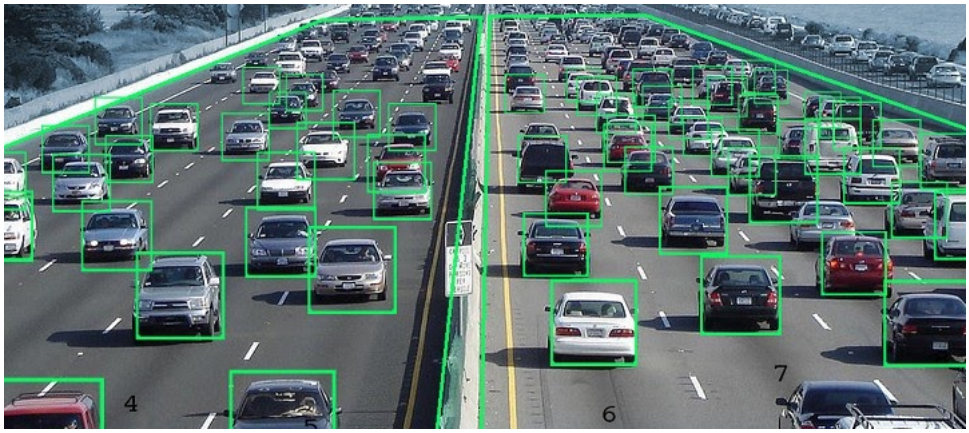


Figure 2.13: Gantry cameras with vehicle detection [Medium, 2018]

There are a number of factors that contribute to the latency of the camera data. These factors include the camera stream latency and the processing time of the footage. If the footage is processed on site then the latency of the processing and the stream of the metric data would be factors contributing to the latency. Electroautomation roadside cameras have options for full internet capabilities, and allow a real-time live feed of the camera footage to be viewed on a standard web browser [Electro Automation, 2021b].

TII has traffic camera feeds available online. These provide a snapshot of the traffic in specific locations. Generally these snapshots are around 10 minutes behind real time. However it may be possible to obtain this data directly from TII, and also to gain the live feed rather than just snapshots. This would allow us to extract information from the feed such as what lane vehicles are in, the vehicles speed and direction of travel, and the class of vehicle. Machine learning algorithms are used to extract this data by producing vehicle counts of the moving objects [Gibbs, 2020]. The data stream that we may have available to our digital twin model would likely be this processed data of the vehicle counts. According to [Song, 2019], the vehicle detection accuracy from such systems is thought to be around 92%-93%. For a 30fps camera, speed error rate is around 2% for vehicles travelling in the range of 70 km/h–100 km/h [Javadi et al., 2019].

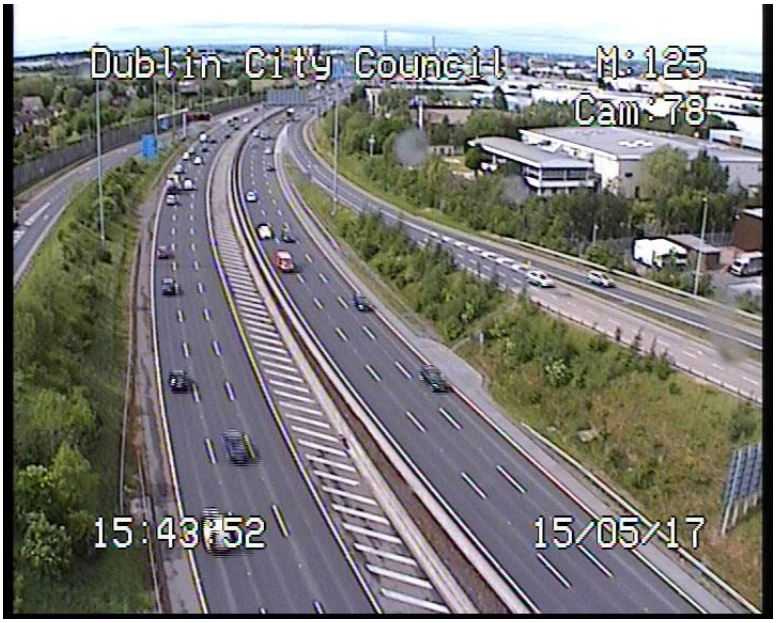


Figure 2.14: Roadside camera on M50 at Ballymount exit [Dublin City Council, 2017]

Drone Cameras

There have been tests using drones to monitor traffic flow. Using a camera mounted to the drone and computer vision techniques it is possible for drones to return the number of vehicles, the vehicle type, their direction of travel, the location of a vehicle, and a timestamp. However a major disadvantage to drones is the short battery life. Typically drones can only fly for around 15 minutes on 1 charge. During a study conducted in Canada [Sowers, 2017], 4 battery packs were used to fly drones for 1 hour at a time to monitor traffic. This would not be a viable solution for continuous traffic monitoring to gather data to build a digital twin.

Figure 2.15: Drone monitoring traffic flow on roundabout [Sowers, 2017]

An advantage of drones for traffic monitoring is that they are non intrusive. They cause no disruption to traffic and drones are typically silent when they are in the air. The maintenance of a drone can be carried out anywhere and everywhere, with no need to ever go to the roadside.

The latency on drones will entirely depend on the video processing done. The typical latency for a drone camera feed is in the milliseconds, however this could be considerably longer due to processing and compiling the metrics from the footage.

The data stream for a drone camera could be similar to that of a roadside camera, with processing on the video data producing traffic counts that would be fed to the model.

Toll Bridge Cameras

Toll bridge sensors can provide the location of a vehicle with a timestamp, the type of vehicle, the direction of travel, as well as some other less relevant information such as vehicle registration. There is a toll bridge on the M50. There are 2 types of sensors used to detect vehicles. The toll bridge is capable of sensing RFID sensors located on select car's windshields, however for the majority of cars cameras are used to read the registration of the vehicle. An advantage of this system is that it is already in place and the data is already available. The issue with this data is that it is back filled and is not published in real time. It may be possible to obtain the camera feed data of the M50 toll. This is essentially the same data stream as a regular roadside camera.

2.3.4 Inductive loops

Inductive loops work on a similar principle to that of a metal detector. They measure the change in magnetic field as an object passes over them. This can be used to detect the presence of vehicles, as well as estimate the type of vehicle due to the size of the passing object. Inductive loops can give the vehicle location, type, and in some cases speed. If there is an inductive loop on each lane of traffic we can then tell which lane or direction the vehicle is moving in. If there are two inductive loops placed in close proximity, they can also be used to estimate the speed of a vehicle by calculating the time taken to cross from one loop to another [Sanderson Associates, 2021].

Inductive loops are usually placed inside the road surface, this means they are intrusive to install and maintain as it requires cutting slots into the road surface, which means the work requires lane closures and obstruction to traffic.

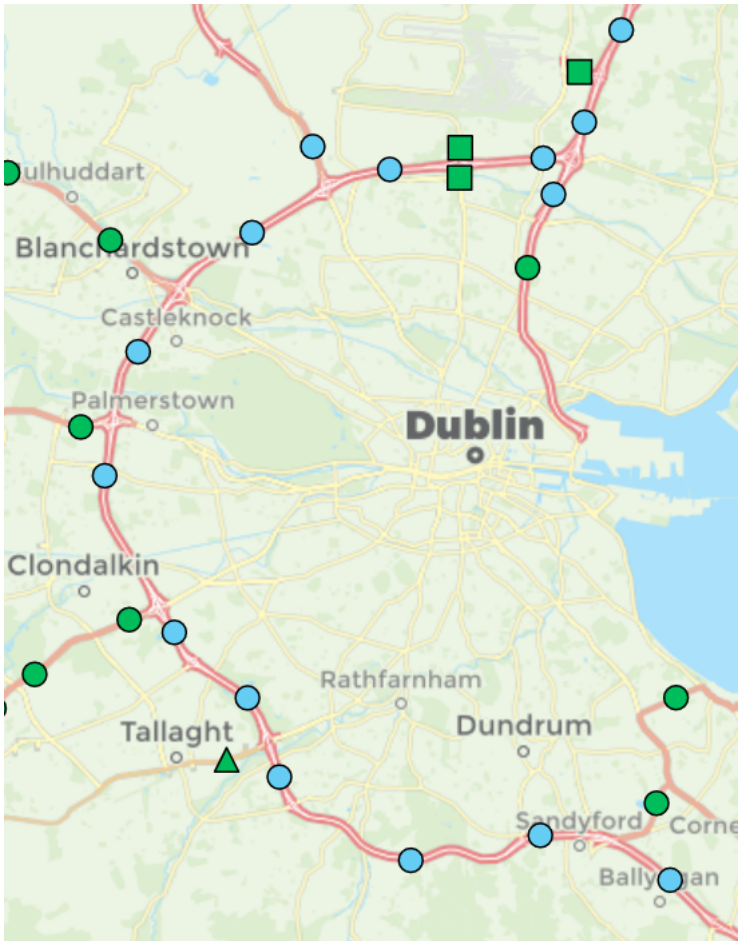


Figure 2.16: Inductive loop locations on the M50 [TII, 2022]

There is a large inductive loop data set for the road networks in Ireland. It provides data with a latency of around 5-10 minutes. The data consists of vehicle location, direction of travel, and vehicle type. This allows us to see the average traffic flow in an area over a 5 minute period, what percentage of vehicles were travelling in a northerly or southerly direction, and what percentage of vehicles were HGVs etc. This data is streamed to the TII website [TII, 2022]. It may be possible to access this data more directly that could improve the latency significantly. Inductive loops are estimated to have a vehicle detection accuracy somewhere between 92% and 98% [Paul Briedis, 2010].

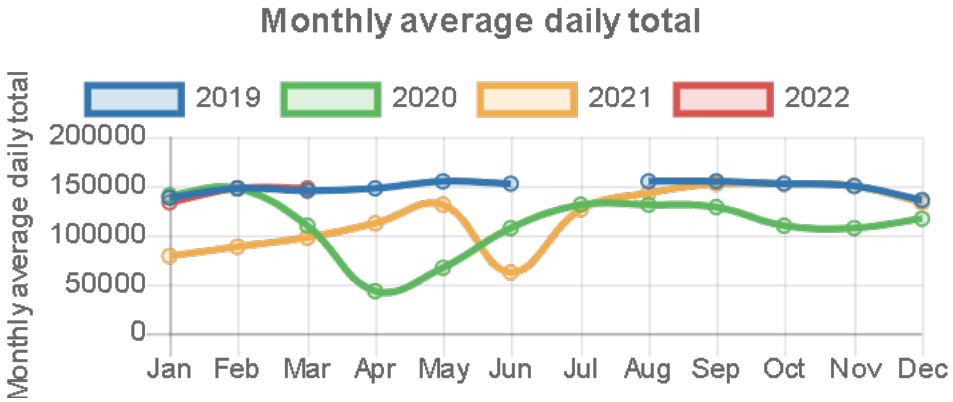


Figure 2.17: Inductive loop traffic volume data [TII, 2022]

2.3.5 Summary of Sensor Research

The sensors that are known to exist on the M50 motorway are inductive loops, the toll bridge, motorway cameras, and vehicles with GPS/GNSS. Table 2.1 outlines a summary of the estimated data properties of these available traffic sensors. The type of data is outlined, this describes the information about an individual vehicle that this sensor can provide. The latency outlines the time delta between an event and the sensor recording that event. The accuracy is the associated accuracy of the sensors observations. The frequency is defined to be the output rate of the sensor i.e. how many measurements the sensor processes per second.

	Type of data	Latency	Accuracy	Frequency
Inductive Loops	Presence, Direction, Lane, Vehicle class	<10ms	Detection: 95%	1HZ
Toll Bridge	Velocity, Lane, Direction, Vehicle class	∼1s	Detection: 93% Velocity: 98%	30HZ
Motorway Cameras	Velocity, Lane, Direction	∼1s	Detection: 93% Velocity: 98%	30HZ
GPS/GNSS	PVT data (position, velocity, time)	<100ms	Position: 3-20 m, Velocity: 0.2kmh	1HZ

Table 2.2: Summary of sensor data properties

2.4 Overview of SUMO

SUMO (Simulation of Urban Mobility) [German Aerospace Center, 2022a] is an open-source traffic simulation software that is capable of simulating large road networks. SUMO is a microscopic traffic simulator (MITSIM). It individually models each vehicle in the network, as opposed to macroscopic traffic simulators that only model the general flow of traffic rather than movements of individual vehicles. Microscopic traffic simulation is a more suitable for our digital twin application as it provides information about individual vehicles rather than the general traffic flow. A simulated version of the M50 is used to evaluate the middleware design for the digital twin.

2.4.1 TraCI

TraCI (Traffic Control Interface) offers a Python control interface that allows for real-time retrieval and modification of data within a SUMO simulation. It uses a TCP based client/server architecture to allow access to SUMO and to control the running of the simulation. TraCI can be used to retrieve live data from a simulation or modify aspects of the simulation itself [German Aerospace Center, 2022b]. TraCI is used to retrieve data from the M50 simulation in order to model virtual sensors that will be used to feed the proposed digital twin.

2.5 Literature Review

It is apparent that digital twins and Intelligent Transport Systems (ITS) are an important area of study in today's world. With regards to digital twins in motorway applications, these can be divided into two main categories. These being digital twins modelling the road surface and topography of the motorway, and digital twins modelling the traffic flow on the motorway, with this review being primarily focused on the latter.

Researchers have agreed that the optimal data sources for the digital twin models for transport systems will come from a variety of sources. These include connected vehicles (CV) as well as roadside units (RSUs). A recent study on a digital twin approach for adaptive traffic control systems (ATCS) found that relying solely on conventional sensor systems such as inductive loop traffic counters a limiting factor in improving the performance of their service, and that great performance improvements can be gained by incorporating CV technology as well as conventional RSUs [Dasgupta et al., 2021]. The idea of leveraging both RSUs as well as CV technology is prevalent in other literature, such as 'A novel digital twin-centric approach for driver intention prediction and traffic congestion avoidance' by A. P. Kumar [Kumar et al., 2018], which states 'Vehicles talk to one another as well as with the roadside IT and electronics equipment to recognize and relay the

real-time situation on the road. The roadside infrastructure also comprises a variety of sensors to measure the distance and the speed of approaching vehicles from every direction'. It is clear that a combination of different data sources will be optimal for feeding the digital twin. These sources could include roadway sensors such as inductive loops, roadside traffic cameras, and toll bridge data, as well as CV technology such as dash-cam data, data from probe vehicles such as GPS enabled taxis and busses, and sat-nav data. Some studies have suggested taking this further and including social media services such as Twitter, and news and weather data services [Du and Chowdhury, 2017].

A common theme among papers is the lack of detail about the methods that may be used to gather data for the digital twin model. In [Dasgupta et al., 2021] the author states that they will use 'trajectory data from connected vehicle (CV) technologies' in order to feed the digital twin for the ATCS. However there are no additional details of how this will be achieved and what kind of sensors will be used. No description of the data format or type of data is provided. This is also apparent in [Kumar et al., 2018] where the author details the types of potential sensors that exist on today's roads, stating 'There are high-fidelity video cameras in plenty along the roads, expressways, tunnels' and that 'wireless access points such as Wi-Fi, 3G, 4G, roadside units, and smart traffic lights have been deployed along the roads'. It is at no point in the paper made clear exactly how this data could be used and processed in order to be able to feed a digital twin model. The process of obtaining data from sensors in real time and transferring this data in order to feed the digital twin is crucial to a real digital twin's success. This is an important gap in the literature this report plans to fill.

There are papers that explore message delivery infrastructures that could be suitable for our digital twin. One such paper is [Du and Chowdhury, 2017]. The paper outlines how Apache Kafka supports the aggregation and processing of raw data from multiple sources. The author in [Du and Chowdhury, 2017] describes the platform as a 'cluster of interconnected computers that rapidly ingest streaming messages generated by data producers (e.g., sensors, roadside units). Different frameworks distribute the data to consumers (e.g., traffic management centers, mobile applications) via publishing (i.e., the data are pushed to the consumers) or consuming (i.e., the consumers pull data from the frameworks' storage space) models.'. In our case the consumer will be the digital twin model, that will pull data from Apache Kafka, which in turn ingests data from the various sensors and CV technology. The author also outlines that the data may come from a 'number of sensors (e.g., sensors in vehicles, cell phones, roadside units)'. The author then states that the 'data will come in varied formats (e.g., PDF, CSV, and structured/unstructured XML)' [Du and Chowdhury, 2017]. This description of the data is missing some information such as the average size of the ingested data, the output frequency of the data (i.e the refresh rate of a particular sensor), and the type of data produced (i.e. position velocity time (PVT) data, relative

position, type of vehicle, presence of vehicle etc). This is information which this report plans to estimate for a number of popular sensors and data sources that may be used to feed the digital twin. It is crucial for the data to be low latency in order for the digital twin to be as near real-time as possible. Tests are conducted in [Du and Chowdhury, 2017] that give insight into how the number of producers and consumers affect the latency times associated with Tp (to producer) and Tc (to consumer), as well as total latency times. This research will be useful when designing the message delivery infrastructure for the digital twin.

Some studies outline the function of a digital twin and why it may be useful, but do not actually create a digital twin. This is evident in [Kumar et al., 2018] where digital twinning is discussed as a concept, outlining the benefits of a digital twin, and how it integrates into their proposed solution. However there is no description of how such a digital twin would be created or what modelling packages would be used to do so. One study that demonstrates an actual digital twin is [Dasgupta et al., 2021]. The digital twin in this study is used to improve the ATCS of an intersection. This digital twin is created in SUMO. It is important to note that this is a simulation of a digital twin environment, rather than a real digital twin, as there is no real time sensor data being used. A key difference between this study and our digital twin is the application. This study is focused on improving the ATCS at an intersection. Our digital twin will be used specifically to model the M50 motorway. While researching the literature, no digital twin (or simulated digital twin) of a motorway or highway was found.

To summarise, it is agreed in the literature that the data sources for digital twins of road networks will come from a variety of sources [Dasgupta et al., 2021] [Kumar et al., 2018]. These sources may include inductive loops, roadside traffic cameras, and toll bridge data, as well as sat-nav data, data from probe vehicles such as GPS enabled taxis and busses, and dash-cam data. There is a lack of literature on how this data is gathered from the sensors, and the data format. It may be possible to use a distributed messaging service for transferring the data from the sensors to the digital twin. An architecture for achieving this is outlined in [Du and Chowdhury, 2017]. This study also outlines performance tests of this framework, which is crucial to the digital twin application as it needs to be low latency. Studies such as [Kumar et al., 2018] describe the concept of a digital twin in traffic situations, and outline why this may be helpful in a number of applications such as feeding a machine learning model. There are also studies that implemented a simulated digital twin in SUMO such as [Dasgupta et al., 2021]. None of these studies however implement a digital twin in a motorway scenario.

3 Methodology & Implementation

3.1 Overview of implementation

The following is designed to be an accurate representation of a middleware architecture which connects motorway sensors and a digital twin of the motorway.

3.1.1 Proposed Architecture

The proposed architecture consists of several key components. These components include the motorway sensors, the digital twin, and the middleware connecting them. In this project, the motorway sensors are simulated using Python. The common source in which the simulated sensors draw their data is a SUMO simulation of the M50 motorway. TraCI is used to interface with the simulation and retrieve traffic data to feed simulated sensors. These simulated sensor-data streams are streamed to the Apache Kafka broker. This broker acts as the main broker for the digital twin. This broker will be hosted on Amazon AWS EC2, which is a scalable cloud computing platform [Amazon Web Services, 2022b]. It allows the user to launch virtual servers known as 'instances'. These instances can be configured to suit your desired CPU, memory, storage, and networking capacity. In order to accurately represent the behaviour of real world traffic data streams, a second Apache Kafka broker is implemented. This broker acts as a 3rd party enterprise server. This server consumes data from several of the simulated data streams. This data is then streamed from the enterprise server to the digital twin broker. This is to account for the fact that the digital twin broker may not have direct access to several 3rd party data streams, such as Google maps [Google, 2021b] data or TomTom [TomTom, 2012] data. It is likely that these data streams will only be accessible from Google or TomTom servers rather than accessing the data directly from the devices. This server will also be hosted on Amazon AWS EC2. The final component of the proposed architecture is the proxy for the digital twin, which is the sole consumer of the digital twin broker.

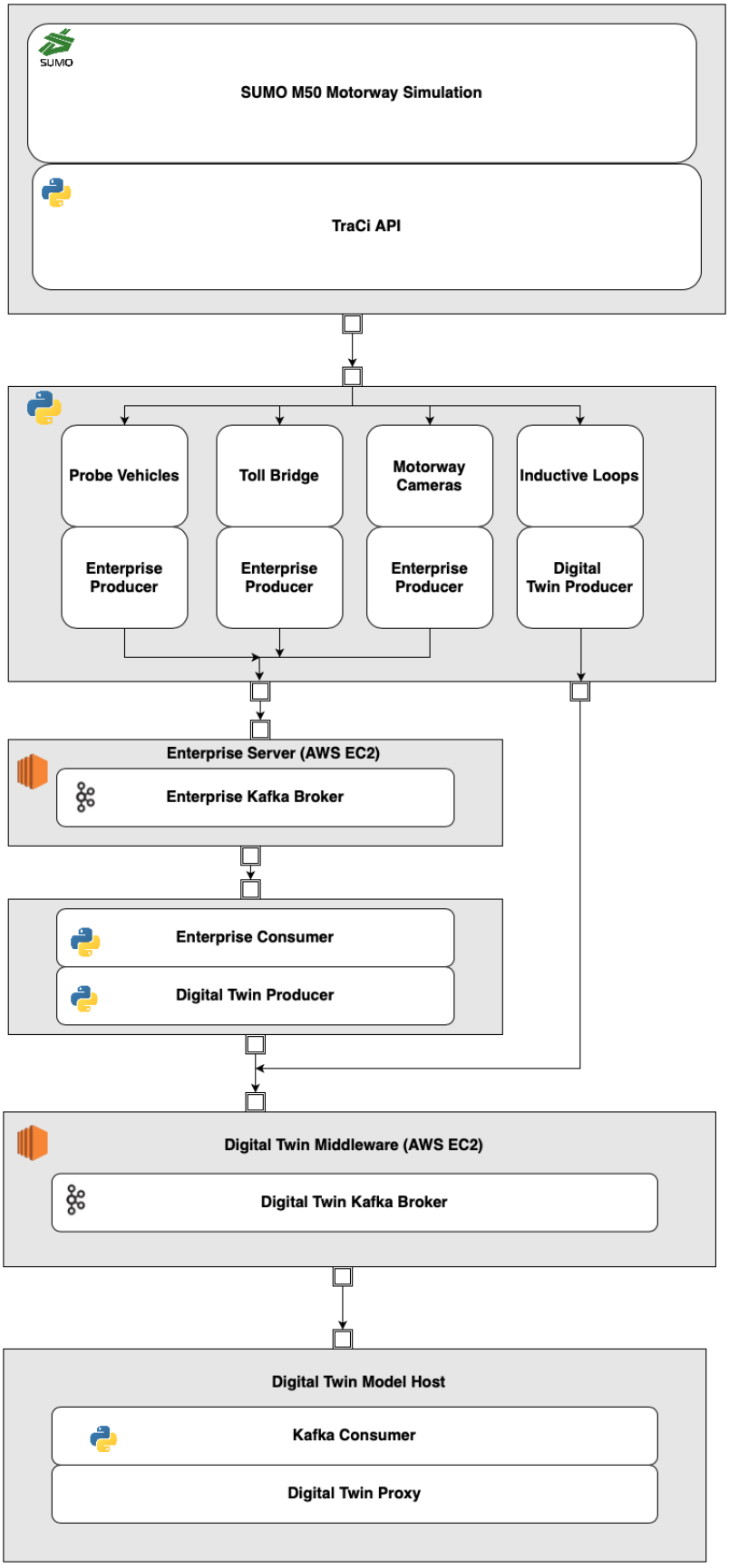


Figure 3.1: Proposed Implementation Architecture

3.2 Motorway Simulation

The source of the data used to drive the virtual sensors is a SUMO simulation of the M50 motorway in Dublin. The simulation source is a paper by Maxime Gueriau and Ivana Dusparic titled *'Quantifying the impact of connected and autonomous vehicles on traffic efficiency and safety in mixed traffic'* [Guériau and Dusparic, 2020]. This paper studies the impact of varying levels of connected autonomous vehicles (CAV) on the flow of traffic. The simulation consists of a 24 hour simulation of a 7km 4-lane stretch of the M50, including two major interchanges with junctions to national roads (N7 and N9).

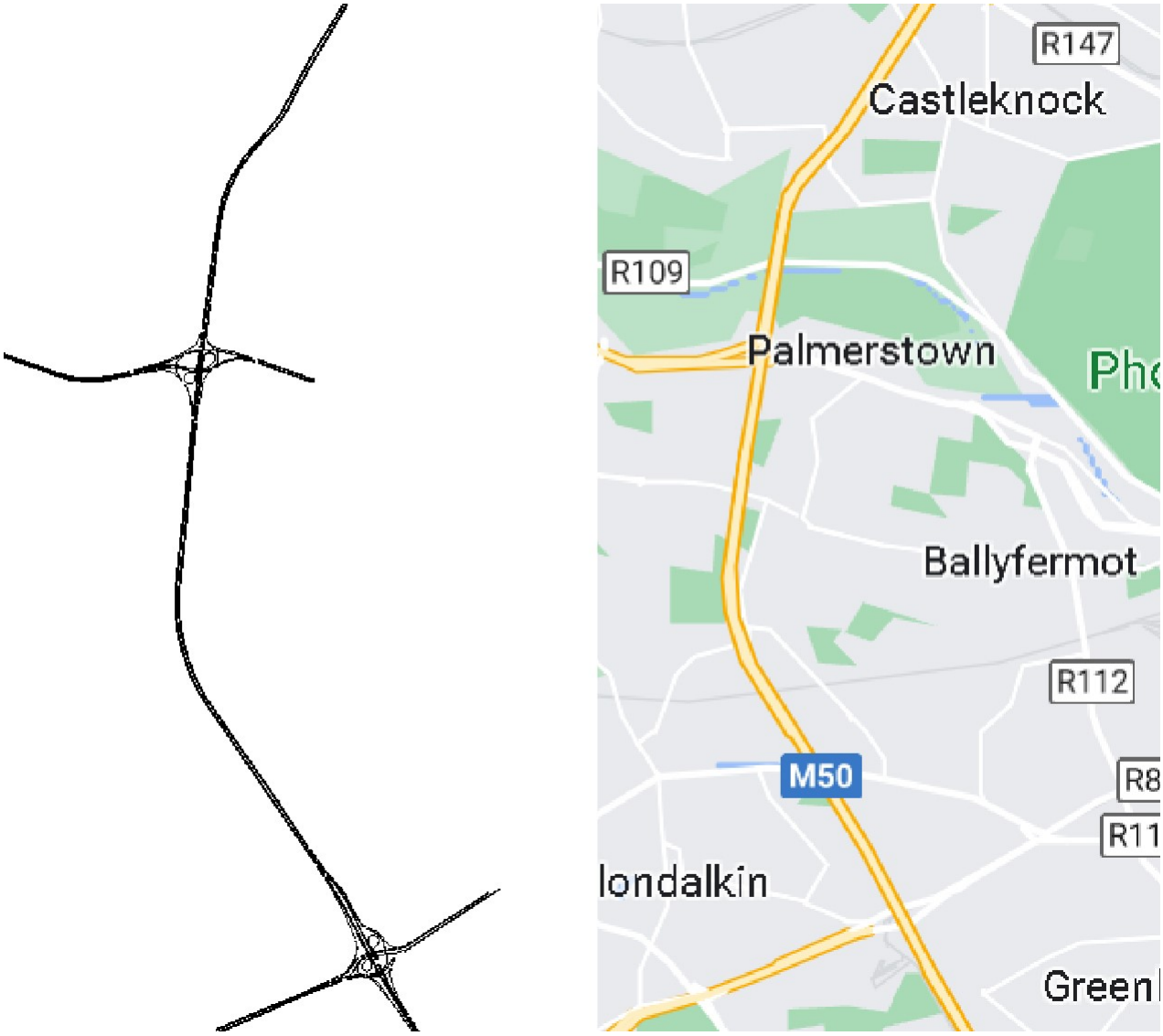


Figure 3.2: Comparing the SUMO simulation (left) to the M50 motorway (right)

3.3 Traffic data retrieval

TraCI is an API which is used to retrieve live data from the SUMO simulation [German Aerospace Center, 2022b]. TraCI implements a client/server architecture in order to access SUMO, where the SUMO simulation acts as the server. The SUMO simulation listens for commands from TraCI on a web port. The TraCI client uses a set of commands to retrieve the desired information. It can retrieve data from individual vehicles in the simulation, as well as data from the road network and traffic detectors in the network, such as inductive loops. Once the required data has been retrieved, the TraCI client uses the 'simulationStep' command to advance the simulation by 1 simulation step. This mean that the simulation does not advance until TraCI receives its data and gives the 'simulationStep' command. The TraCI Python library is used to issue TraCI commands programmatically to the simulation. This allows for automated value retrieval from the simulation using TraCI. The default simulation step size for the M50 motorway simulation is 0.5 seconds.

3.4 Modelling the Sensors

A number of motorway traffic sensors are modelled in order to feed the proposed digital twin. The design of these virtualised sensors is based on the research in section 2.3. TraCI is used to retrieve data from the simulation about the road network and individual vehicles. This data is then compiled and packaged in order to be representative of the data output of the sensors on the road network. This includes modelling the type of data the sensor outputs and the output frequency. Noise is also added to the data in accordance to the estimated accuracy of the sensors.

3.4.1 Inductive Loops

Inductive loops can be implemented natively in SUMO. The M50 simulation has a set of inductive loops already included in its configuration. Values for id, lane, pos, freq, and file are defined depending on the desired design of the induction loop. The id is a unique identifier for the induction loop, lane refers to the lane in which the inductive loop is located, and pos is defined as the position in metres along the road that the inductive loop is situated. Freq is the output frequency to the file output. It is still possible to obtain inductive loop data at any time using TraCI regardless of the configured freq value. File is the output file in which the inductive loop data is written. An example of a 4 lane induction loop configuration is shown below:

```
<inductionLoop id="L1" lane="1" pos="10" freq="1" file="out.xml"/>
<inductionLoop id="L2" lane="2" pos="10" freq="1" file="out.xml"/>
<inductionLoop id="L3" lane="3" pos="10" freq="1" file="out.xml"/>
<inductionLoop id="L4" lane="4" pos="10" freq="1" file="out.xml"/>
```

Figure 3.3: 4 lane induction loop configuration

The data from the inductive loops can be dynamically read during each simulation step using TraCI [German Aerospace Center, 2022c]. The native inductive loops have an output frequency in accordance to the simulations step size. Modelling a different output frequency can be achieved programmatically using Python. During each simulation step TraCI reads the inductive loops vehicle counts. These counts are added to a tally which includes counts from the previous N simulation steps, where N is the number of simulation steps since the inductive loop has streamed data to Kafka. The inductive loop Kafka producers have an output frequency of 1 second. For a simulation step size of 0.5, the sensors vehicle count tally includes the vehicle counts from the previous N=2 simulation steps. It is possible for the sensor to detect more than 1 unique vehicle per simulation step. It is also technically possible to return a count of 2 for a single vehicle, however unlikely unless the vehicle is at rest. This tally is then streamed to Kafka using the inductive loop producers and the tally is reset.

In order to accurately represent real inductive loop data streams, noise is added to the data. The estimated vehicle detection accuracy of inductive loops is between 92% and 98% [Paul Briedis, 2010]. The mean estimated accuracy is therefore in the region of 95%. This accuracy will be reflected in our simulated inductive loop data streams. This is achieved by incorporating a spurious vehicle detection data point every 0.05 data points.

Each virtual inductive loop sensor contains a set of inductive loops. These sets of inductive loops are loops which exist on the same road network but on separate lanes. Since the M50 has 4 lanes there is 4 inductive loops per set.

The sensor output format for the inductive loops can be seen below:

```
data = {
    'loop_id': string,
    'lane_1': string,
    'lane_2': string,
    'lane_3': string,
    'lane_4': string,
    'timestamp': string
}
```

Figure 3.4: Inductive loop sensor output

- **loop_id:** Unique identifier for the set of inductive loops.
- **lane_1:** Vehicle count for the inductive loop on lane 1.
- **lane_2:** Vehicle count for the inductive loop on lane 2.
- **lane_3:** Vehicle count for the inductive loop on lane 3.
- **lane_4:** Vehicle count for the inductive loop on lane 4.
- **timestamp:** The simulation timestamp the measurements were taken. This is independent of the true time outside of the simulation.

3.4.2 Motorway Cameras

Motorway cameras are mounted on gantries in various locations on the M50 motorway. The locations of these cameras can be found on the TII website [TII, 2022]. In order to obtain meaningful information from the camera streams, additional processing is necessary. It is possible to extract information from the camera feeds using techniques described in [Yu, 2018]. This information may include the vehicle's distance to the camera, the vehicle's speed, and which lane the vehicle is in, and the direction of travel.

Unlike inductive loops, there is no support for motorway cameras built into SUMO. The camera feeds are modelled using TraCI and Python. The modelled cameras are situated in the same locations the real cameras are situated. These locations are recorded as coordinates. TraCI is used to retrieve data about all vehicles within a 200m radius of the cameras coordinates. A radius of 200m is estimated based on the live feed of the cameras available at [TII, 2022]. Depending on the orientation of the camera, vehicles that are 'behind' the camera are ignored. This orientation is based on the orientation of the real cameras on the M50. The real cameras are orientated in arbitrary directions on the

motorway. The virtualised camera's orientation is simplified to north or south facing. This greatly simplifies the decision boundary of vehicles that are ignored by the camera.

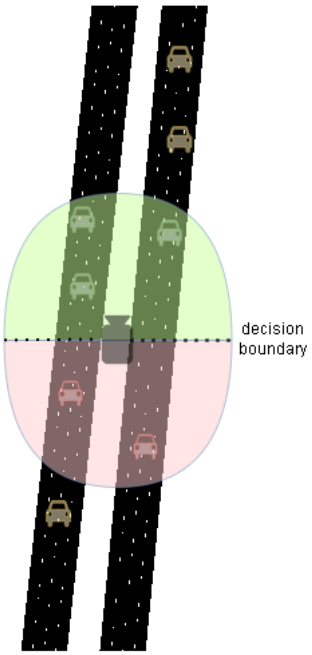


Figure 3.5: FOV of virtualised camera with 3 vehicles in detection zone

The vehicle id's of all vehicles which are hypothetically within the cameras field of view (FOV) are returned. For each vehicle in the cameras FOV, the speed, distance to the virtual camera, lane, lane index, and direction of travel are recorded.

To accurately represent the limitations of the sensor, noise is added to each data point. As outlined in [Song, 2019], the detection accuracy is estimated between 92% and 98%. The speed measurement is estimated to have an error rate of 2% using a 30 fps camera [Javadi et al., 2019]. A distance error of 5% is also added to each data point.

The data format of the motorway camera data can be seen below in Figure 3.6.

```
data = {
    'camera_id': string,
    'lane_id': string,
    'lane_index': string,
    'direction': string,
    'distance': string,
    'speed': string,
    'timestamp': string
}
```

Figure 3.6: Camera sensor output

- **camera_id:** Unique identifier for the camera.
- **lane_id:** Which road the vehicle is travelling on.
- **lane_index:** Which lane on the road the vehicle is in.
- **direction:** The direction the vehicle is travelling.
- **distance:** The distance between the vehicle and the camera.
- **speed:** The speed in kilometres per hour the vehicle is travelling.
- **timestamp:** The simulation timestamp the measurements were taken. This is independent of the true time outside of the simulation.

3.4.3 Toll Bridge

The toll bridge cameras are modelled akin to the regular motorway cameras. The toll bridge cameras are located in the same location as the toll bridge cameras on the M50. A similar method as outlined in Figure 3.5 is used to calculate the vehicles which are in view of the cameras. Unlike the regular motorway cameras, the toll bridge sensors also return the vehicle class of the vehicles in it's FOV. This is similar to how the real toll bridge cameras operate on the M50. The speed, distance to the virtual camera, lane index, direction of travel, and vehicle class is recorded for each vehicle in the sensor's FOV. The same error rates are added to each data point as outlined in 3.4.2.

The data format of the toll-bridge data can be seen below in Figure 3.7.

```
data = {
    'lane_id': string,
    'lane_index': string,
    'direction': string,
    'distance': string,
    'speed': string,
    'class': string,
    'timestamp': string
}
```

Figure 3.7: Toll bridge sensor output

- **lane_id:** Which road the vehicle is travelling on.
- **lane_index:** Which lane on the road the vehicle is in.
- **direction:** The direction the vehicle is travelling.
- **distance:** The distance between the vehicle and the toll bridge.

- **speed:** The speed in kilometres per hour the vehicle is travelling.
- **class:** The class of the vehicle.
- **timestamp:** The simulation timestamp the measurements were taken. This is independent of the true time outside of the simulation.

3.4.4 Probe Vehicles

Probe vehicle data can come from a number of sources. These may include real-time bus tracking, Google maps data, sat-nav data, and other sources. These probe vehicles may all produce slightly different types of data depending on the source, however it will be some form of GPS data.

Some generalisations are made in order to practically implement the probe vehicle sensors:

- All probe vehicles produce the same GPS style data format.
- 20% of all vehicles in the simulation are treated as probe vehicles.
- Probe vehicles share a common producer. In reality each 3rd party data source would be a producer.

The frequency output of the probe vehicle sensors is estimated be to be 1HZ. The location of a single probe vehicle is calculated using the TraCI command:

```
x, y = TraCI.vehicle.getPosition(vehID)
```

This returns the vehicles x,y location in Cartesian coordinates. In order to accurately reflect real GPS data, the coordinates are converted into latitude and longitudinal coordinates using the following TraCI command:

```
lat, lon = TraCI.simulation.convertGeo(x,y)
```

Where x and y are the previously obtained Cartesian coordinates.

Each GPS measurement from a probe vehicle will have inherent noise. GPS is accurate for 3 metres for devices such as the Garmin 16x GPS device[Garmin, 2008], and up to 20 metres for services such as Google maps [Google, 2021b]. An error of up to 10 metres is estimated for the probe vehicle sensors. This noise is implemented by superimposing a circle centered on the coordinates of the probe vehicle, using a random variable within the circle to represent the location measurement with added noise. The longitude error and latitude error is calculated and the probe vehicle location is returned with this error:

```
r  = 10/111300 #converts 10 metres into degrees
w = r * math.sqrt(random.uniform(0, 1))
t = 2 * math.pi * (random.uniform(0, 1))
x = w * math.cos(t)
lon_error = x/math.cos(lat_true)
lat_error = w * math.sin(t)

coordinates = (lon_true + lon_error, lat_true + lat_error)
```

Figure 3.8: Probe vehicle location error calculation

The speed measurement of GPS is estimated to be around 0.2 km/h [US Government, 2021]. This inaccuracy is also included to be reflected in the data streams.

The data format of the probe vehicle data can be seen in Figure 3.9.

```
data = {
    'probe_id': string,
    'location': string,
    'speed': string,
    'vehicle type': string,
    'timestamp': string
}
```

Figure 3.9: Probe vehicle sensor output

- **probe_id:** A unique identifier for the vehicle.
- **location:** The latitude and longitudinal location of the vehicle.
- **speed:** The speed in kilometres per hour the vehicle is travelling.
- **vehicle type:** The type of vehicle.
- **timestamp:** The simulation timestamp the measurements were taken. This is independent of the true time outside of the simulation.

3.5 Kafka implementation

As explained in section 2.2.5, Apache Kafka is the technology chosen for the communication architecture between the sensors and the digital twin. The Kafka implementation consists of 3 key components.

3.5.1 Brokers

Kafka is run as a cluster of one or more servers. These servers are known as brokers. The brokers are the endpoints to which the producers write data and the consumers read data. Brokers are responsible for handling incoming requests, managing the topics and partitions, and handling the replication of messages.

Topics

A Kafka topic is created for each sensor group. Inside each topic there are several partitions.

- Inductive loops: The inductive loop topic contains a partition for each group of inductive loops.
- Toll bridge: The toll bridge topic has a partition for northbound traffic and a partition for southbound traffic.
- Motorway cameras: The motorway cameras topic has a partition for each motorway camera location.
- Probe vehicles: The probe vehicles topic has a partition for each 3rd party data source.

Figure 3.11 outlines the layout inside the Kafka broker detailing the potential partitions per topic split:

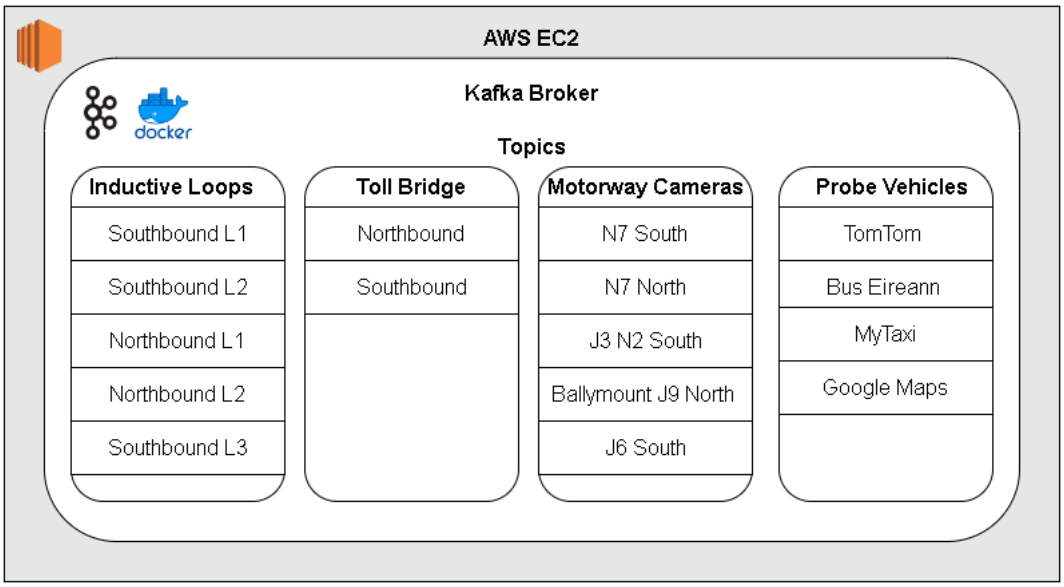


Figure 3.10: Inside the Kafka broker

3.5.2 Producers

The producers are publishers responsible for writing messages to Kafka topics. In this system, the sensors act as producers. The producers take the sensor data outlined in section 3.4 and stream this data to the Kafka cluster.

The specified 'bootstrap_servers' of a Kafka producer indicates the endpoint or list of endpoint(s) of the Kafka broker(s). For a producer publishing messages to a single broker, the 'broker_endpoint' is the broker's endpoint. For a producer publishing messages to multiple Kafka brokers, then the 'broker_endpoint' is a list of Kafka broker endpoints.

The send method of the KafkaProducer class is used to publish messages to the 'bootstrap_servers'. The send method takes the message as an argument. It also takes the topic and partition in which to publish the message as an argument.

```
from kafka import KafkaProducer

producer = KafkaProducer(bootstrap_servers=${broker_endpoint})

producer.send(topic=topic, value=message, partition=partition)
```

Figure 3.11: Kafka Python producer code

3.5.3 Consumers

The consumers are responsible for reading records from the Kafka brokers. Consumers can subscribe to messages from specific topics. In this system, the sole Kafka consumer is the digital twin proxy. The consumer specifies the topics to be consumed and the broker's endpoint.

Similar to the producers, the specified 'bootstrap_servers' of a Kafka consumer indicates the endpoint or list of endpoint(s) of the Kafka broker(s). Consumers publishing messages to a single broker, the 'broker_endpoint' is the broker's endpoint. Consumers publishing messages to multiple Kafka brokers, then the 'broker_endpoint' is a list of Kafka broker endpoints.

The consumer subscribes to 'topics'. This can be a singular topic or a list of topics. The consumer will consume messages from these topics on the specified 'bootstrap_servers'. These messages are consumed sequentially in a for loop. During each iteration, a single message can be operated on. In this project, the messages are written to a text file depending on which topic and partition the message originates from.

```
from kafka import KafkaConsumer

consumer = KafkaConsumer(bootstrap_servers=${broker_endpoint})

consumer.subscribe(topics=${topics})

for msg in consumer:
    #calculate the latency of the msg
    #write the message to text file depending on it's topic and partition
```

Figure 3.12: Kafka Python consumer code

3.6 Deployment

3.6.1 Brokers

The broker(s) are deployed in Docker containers [Docker, 2022a] on remote servers hosted by AWS known as EC2 instances [Amazon Web Services, 2022b].

Docker

Docker is a framework for building and managing containers. The Kafka broker's configuration is defined within the docker 'compose' file. The docker compose file can then be used to deploy that specific Kafka broker. This allows for brokers to be deployed to docker containers in a repeatable manner [Docker, 2022b].

An example docker compose file for deploying a Kafka broker can be seen below. The host server's local ipv4 address is defined as 'local_ipv4', and the Zookeeper host's ipv4 is defined as 'zookeeper_ipv4'. The 'local_ipv4' is the endpoint used by the Kafka producers and consumers to publish and subscribe to messages.

```
kafka:
  image: ${kafka_version}
  container_name: 'kafka'
  ports:
    - '9094:9094'
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://${public_ipv4}:9094
    KAFKA_ZOOKEEPER_CONNECT: ${zookeeper_ipv4}:2181
    ALLOW_PLAINTEXT_LISTENER: 'yes'
```

Figure 3.13: Docker compose Kafka container

AWS EC2

AWS EC2 is an elastic cloud computing platform. It allows for the deployment of remote servers with a choice of processor, storage, network, and operating system. The servers, known as 'instances', can be rented on a pay-as-you-go basis [Amazon Web Services, 2022b]. The Kafka brokers are hosted on EC2 instances.

3.6.2 Producers and Consumers

The producers and consumers are running on the local machine specified in 4.2. There are 3 files which contain producers and/or consumers. The file responsible for interfacing with the simulation using TraCI is called simulation.py. This script contains the producers for the virtual sensors. Depending on the particular sensor group, these producers may produce to the digital twin Kafka cluster, or the enterprise Kafka cluster. The enterprise_server.py script contains the enterprise consumers as well as the digital twin producers for the data which is sent to the enterprise server and not directly to the digital twin consumers. The digital_twin_proxy.py file contains the digital twin consumer(s).

The file structure for the producer and consumer scripts can be seen in Figure 3.14.

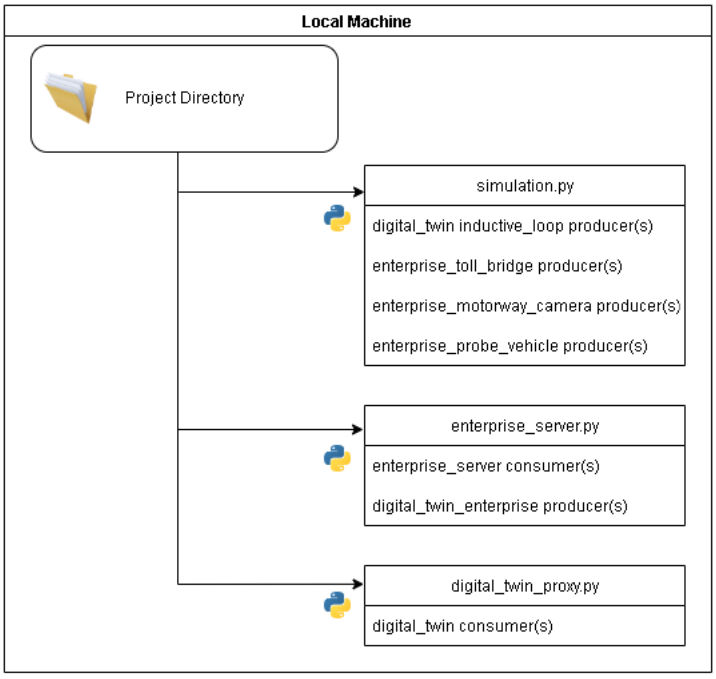


Figure 3.14: Project file structure

4 Evaluation

In this section the performance of the communication architecture is examined. The performance of the system is analysed under several simulation configurations. The SUMO M50 simulation outlined in section 3.2 is used as a common source of traffic data. This traffic data is used to drive the virtual sensors outlined in section 3.4. The virtual sensors act as Kafka producers for the communication architecture. The digital twin acts as the consumer.

4.1 Testing metrics

There are a number of key metrics which are used to evaluate the performance of the communication architecture. They are as follows:

Throughput: Throughput is defined as the amount of information a system can produce and deliver to the consumer in a given period of time. In the case of Kafka, the throughput is the number of messages per second that passes through the Kafka cluster. The throughput of the system is a critical component of the system as it determines how many messages the system can process per second. If the throughput is too low, the system will not be able to process messages fast enough, which will inevitably lead to a backlog and increased latency.

Latency: Latency is defined as the time delay for data transfer between one point and another. For this system, the latency is the time taken to send from producer to consumer. Latency is an important metric as high latency will lead to an increased time delta between the motorway sensors and the digital twin. It is therefore extremely important to minimise latency in order to optimise the performance of the digital twin.

Availability: Availability is defined as the ability for a service to offer the same performance in spite of a failure. For this architecture this means the ability for the service to guarantee message delivery to the digital twin in spite of a broker failure.

4.1.1 Test parameters

4.1.2 Configurable parameters

There are several configurable parameters which can influence the performance of the communication architecture. These may include:

- **Number of messages:** The number of messages produced is a function of the volume of vehicles in the simulation. There will be a larger volume of messages during busy periods in the simulation compared to quieter periods. The system will be tested for both a large and small volume of messages. Simulations during 00:00-01:00 are used to simulate quiet time periods with a small number of messages, and simulations during 09:00-10:00 are used to simulate busy rush hour traffic with a large number of messages being produced.

  Table 4.1 shows the number of messages produced by each topic during high and low traffic simulations.

	loop_messages	cam_messages	toll_messages	probe_messages
Low traffic	7200	48688	5691	135605
High traffic	7200	408459	64952	1087701

Table 4.1: Messages by topic

  The inductive loop sensors produce the same number of messages regardless of traffic volume. However the motorway cameras and toll bridge sensors produce data proportional to the number of vehicles in their detection zones. The increase in traffic also increases the number of probe vehicles on the road, which leads to an increased volume of messages in the topic.

- **Broker hardware specifications:** The hardware specifications of the servers in which the Kafka broker(s) run will have an impact on the performance of the system. The system is tested using various hardware with varying amounts of memory and processing power.

- **Number of consumers:** The number of consumers may affect the performance of the system. The system is tested using a varying number of consumers in order to assess the difference in performance between them.

- **Number of brokers:** The system is tested using various configurations. Several tests are conducted using differing numbers of brokers in the cluster.

- **Compression algorithm:** Compression will decrease the size of messages sent from the producer, however additional processing time may be needed to compress and decompress the data. The gzip compression algorithm is tested, as well as no compression.
- **Serialisation method:** In order to transfer the data from the producer to Kafka, the data needs to be serialised. The data is serialised using UTF-8 encoding.
- **Replication factor:** Replication in Kafka means that multiple copies of a message are replicated in different brokers. Higher replication is desirable as it leads to increased availability, however it may hinder throughput and latency. The effects of replication on these metrics are tested.
- **Message batching:** Kafka has the ability to batch messages which are produced in a short time window and send them in one request. Batching can significantly increase the throughput of the system, however too much batching can lead to increased latency between a measurement and the batch containing that measurement being sent by the producer.
- **Distance to brokers:** The distance from producer to broker and consumer to broker will effect the latency of the system. It is important to understand the effects of relative broker placement on the performance of the system.

4.1.3 Fixed parameters

- **Network speed:** The network speed will have a significant affect on the performance of the system. It is important to minimise the effects of the network speed across test cases in order to obtain consistent results. All tests are performed on the same network in the same location unless otherwise stated.
- **Enterprise server latency:** The end-to-end latency between the sensors and the enterprise server influences the overall latency of the digital twin. It is assumed that the performance of the 3rd party server is fixed. The 3rd party server also uses Apache Kafka as its communication architecture, however the configuration of this server is fixed during all tests.
- **Simulation host machine:** The hardware in which the simulation is running will affect the speed of the simulation. The speed of the simulation will influence the throughput of the sensors. A more powerful machine which is capable of running the simulation more quickly will lead to more messages being produced per second. It is important to note that the simulation time is not necessarily the same as the simulated time.

- **Limitations of the simulation:** In ideal conditions, there is a Kafka producer for each RSU. These producers would run independently of each other, allowing multiple producers to produce messages to Kafka simultaneously. However it is impractical to implement this behaviour with a single host machine running the SUMO simulation. The simulation therefore must produce messages sequentially rather than in parallel.

4.2 Test hardware

There are a number of hardware choices which are used. With regards to the Kafka brokers, there are two EC2 instances which are used. The specifications of these instances are detailed below:

	Linux/UNIX t2.large	**Linux/UNIX t2.xlarge**
vCPU	2	4
Memory	8GB	16GB
Network performance	low-moderate	moderate

Table 4.2: Test hardware comparison

The SUMO simulation, the Kafka producers, the enterprise server, and the Kafka consumers run on a local machine. The specifications of this machine are outlined below:

- OS: WINDOWS 10
- Processor: i5-9600k @ 3.70GHZ
- Memory: 32GB RAM

4.3 Testing methodology

There are two types of tests which are conducted for each testing configuration. These tests are outlined below:

4.3.1 Simulation testing

Testing using the SUMO simulation. Each configuration is conducted for simulation times between 00:00-01:00 to simulate low volume situations, and 09:00-10:00 to simulate high traffic situations.

4.3.2 Benchmark testing

Each configuration is benchmarked using a data set containing 1 million probe vehicle data points. These data points are produced by the producer as quickly as possible, and consumed by the consumer quickly as possible.

4.3.3 Latency measurements

For each message, the epoch time directly before the message is produced by the Kafka producer is recorded. The epoch time directly after a message is consumed by the digital twin consumer is also recorded. The end-to-end latency is calculated as the time difference between the consumed time and the produced time of an individual message. This measurement will vary between the sensor groups. The inductive loop sensors are directly streamed from the RSUs to the digital twin Kafka cluster. This indicates 2 network hops are required to reach the digital twin proxy. The toll bridge cameras, motorway cameras, and the probe vehicle measurements are streamed to an enterprise broker from the sensors. These measurements are then consumed by the enterprise server and streamed to the digital twin Kafka cluster. They require 4 network hops in order to reach the digital twin proxy.

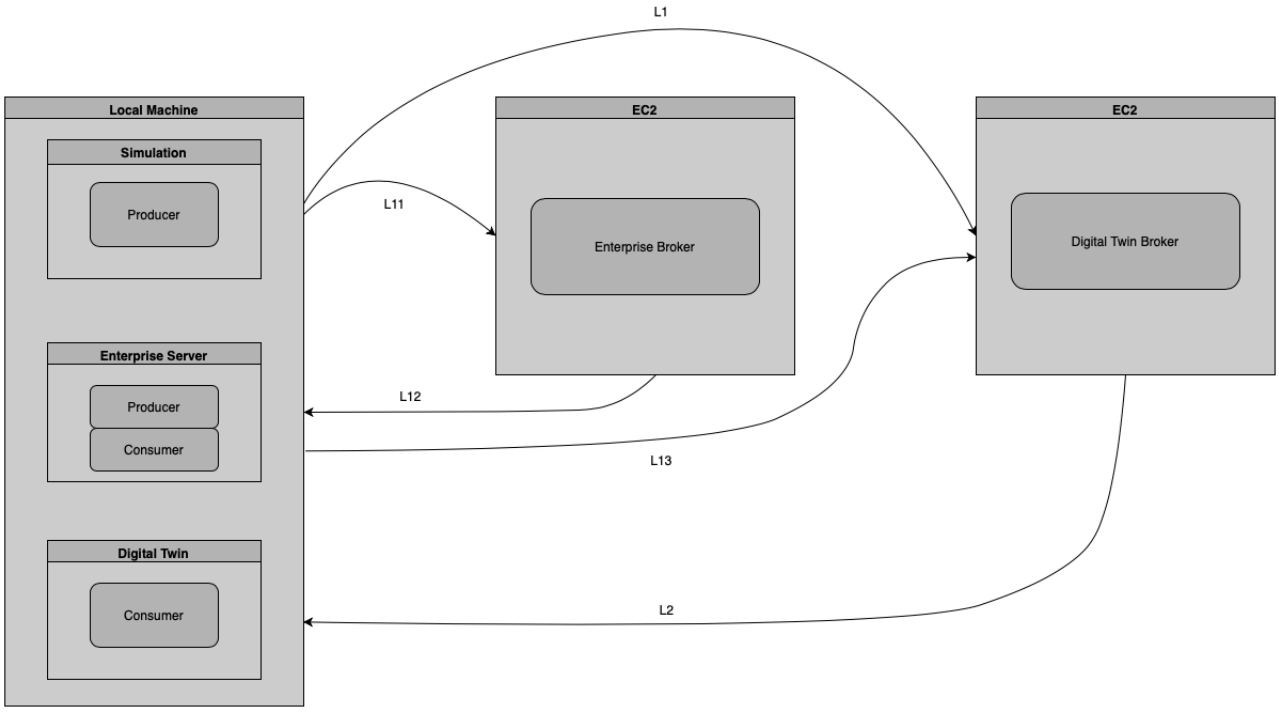


Figure 4.1: Latency measurements overview

The end to end latency of the inductive loop sensors is defined as:

$$\Delta t = L1 + L2 \tag{1}$$

The end to end latency of the toll bridge cameras, motorway cameras, and probe vehicles is defined as:

$$\Delta t = L11 + L12 + L13 + L2 \tag{2}$$

The median latency per sensor group is recorded for both low and high volume simulations.

4.3.4 Throughput measurements

The throughput is measured to be the total number of messages produced and consumed by the system divided by the total simulation run time.

4.3.5 Default Kafka configuration

- The Kafka version used during all tests is Kafka 2.5.0 [Apache Kafka, 2020].
- Zookeeper version 3.8.0 is used for all tests.
- During all tests each instance of Kafka and Zookeeper are running on their own dedicated hardware. This means that if a configuration states 3 Kafka brokers running on t2.large EC2 instances, this means that each Kafka broker has its own t2.large instance.
- The serialisation encoding and decoding methods used for all configurations is the Python json UTF-8 encoding module [Python.org, 2022].
- The default batch size defined in the Python Kafka module is 16384 bytes.
- The default number of partitions used for each topic is defined below unless otherwise stated. There is a single partition for both north and south inductive loops. There is 8 motorway camera partitions, one for each individual camera. There are 2 toll bridge partitions, one partition for northbound traffic and one for southbound traffic. There are 10 partitions for the probe vehicles. A single probe vehicle partition is used to represent a single probe vehicle data source, such as 1 partition for Google maps data, 1 partition for TomTom data, 1 partition for Garmin data, and so forth. The probe vehicles have 10 partitions. This figure is selected as an estimated number of 3rd party data sources which may be available. Each probe vehicle is assigned a random

partition to simplify the implementation, however in reality probe vehicles would be assigned to specific partitions depending on the 3rd party data source.

	Inductive loops	Motorway cameras	Toll bridge cameras	Probe vehicles
#Partitions	2	8	2	10

Table 4.3: Default number of partitions per topic

- The enterprise Kafka broker remains consistent throughout all configurations. It runs a single broker on a t2.large EC2 instance. The producers which produce directly to the enterprise broker run default batch sizing with no compression. Each topic on the enterprise broker has a single partition.

4.4 Test configurations

The following section outlines the configurations under test.

Config	Hardware	#Brokers	#Consumers	Compression	Batching	Replication
A	t2.large	1	1	None	Default	1
B	t2.large	1	1	gzip	Default	1
C	t2.large	1	1	None	2*Default	1
D	t2.large	1	3	gzip	Default	1
E	t2.xlarge	1	1	gzip	Default	1
F	t2.large	3	1	gzip	Default	3
G	t2.large	3	3	gzip	Default	3

Table 4.4: Simulation test configurations

- **Hardware:** The two hardware specifications under test are the AWS EC2 t2.large and t2.xlarge.
- **#Brokers:** The performance of a single broker Kafka cluster is compared to that of a 3 broker Kafka cluster.
- **#Consumers:** The system is tested using both a single consumer and 3 consumers running in parallel. Each message is only consumed by a single consumer, meaning a single message can't be consumed more than once.
- **Compression:** The effects of compression on the performance of the system are tested using both uncompressed configurations and configurations using the gzip compression algorithm.

- **Batching:** The default batch size of the producer is doubled in order to analyse the effects of batching on the performance of the system. The default batch size is 16384 bytes.
- **Replication:** Single broker clusters have a replication factor of 1, meaning that the messages are only stored on the single broker. The 3 broker cluster configurations have a replication factor of 3, meaning each message is replicated on all 3 brokers.

4.5 Results

The following section describes the results of the tests conducted using configuration A through G as described in table 4.4. All test configurations are simulated in both high and low traffic situations as described in section 4.3.1, as well as benchmark testing as described in section 4.3.2.

4.5.1 Simulation Results

All simulations are conducted using the architecture outlined in 3.1. The latency measurements that appear in the following section is the median end-to-end latency. The error-bars of all measurements represent the interquartile range (IQR) of each measurment.

Latency by configuration

The following section displays the latency measurements of each configuration during high and low traffic simulations conducted using the architecture outlined in 3.1. The median latency of each configuration is calculated for all messages regardless of topic.

It is worth noting that topics which contain a larger volume of messages will influence the overall latency of the system more so than topics with lower volumes of messages. Table 4.1 shows that during low traffic simulations, approximately 135,000 probe vehicle messages are produced, while only 7200 inductive loop messages are produced. It is clear that the probe vehicle topic will have a greater influence on the latency of the system than the inductive loop topic.

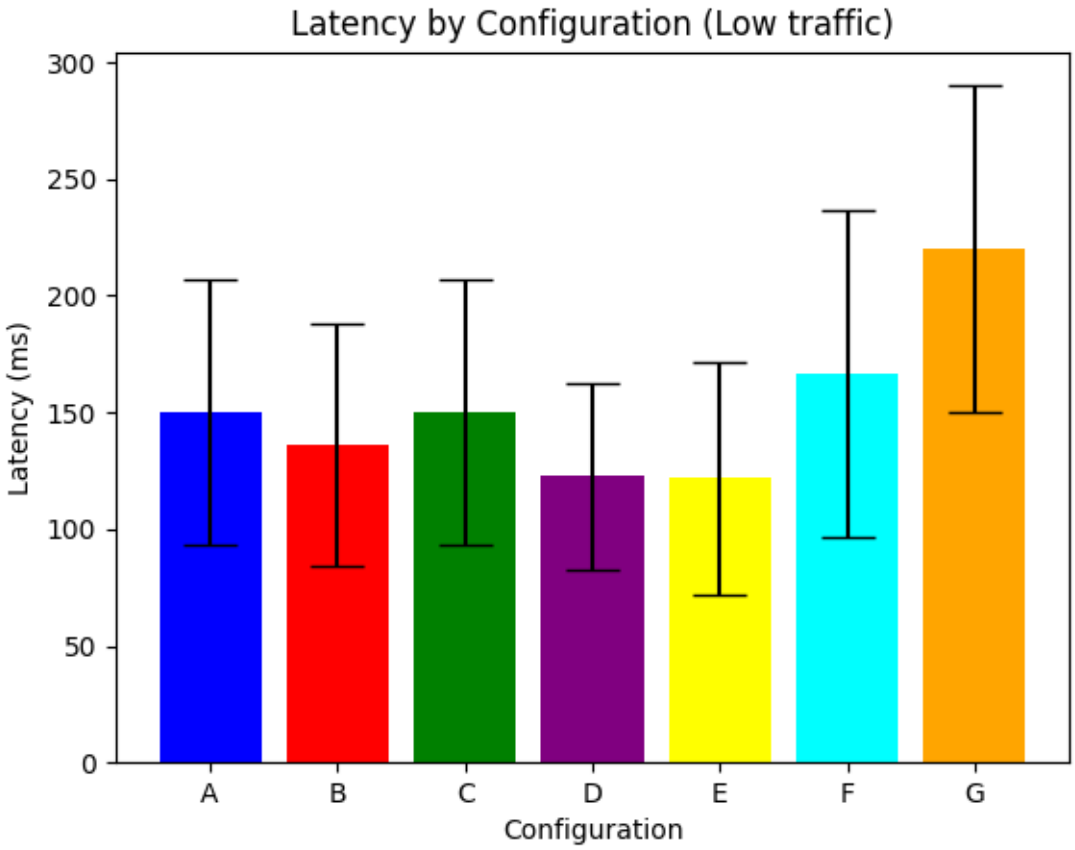


Figure 4.2: latency (low traffic)

Figure 4.2 shows the latency for each configuration in low traffic situations. It can be seen that configurations E and D had the highest performance, with latency of approximately 120 ms. Configuration G had the highest latency with a median measurement around 220 ms. There is an 83% increase in latency between configurations D and E and configuration G. Configurations D and E are single broker configurations, while configuration G is a 3 broker configuration. The overhead of having 3 brokers with replication across the brokers has an increased network overhead. This is likely the principal factor in the increased latency observed in configuration G.

The error bars among all configurations appear to be loosely proportional to the latency of the configuration. That meaning that configurations which experience higher latency have a slightly larger interquartile range. However there does not appear to be a particular configuration that experiences a significantly higher spread.

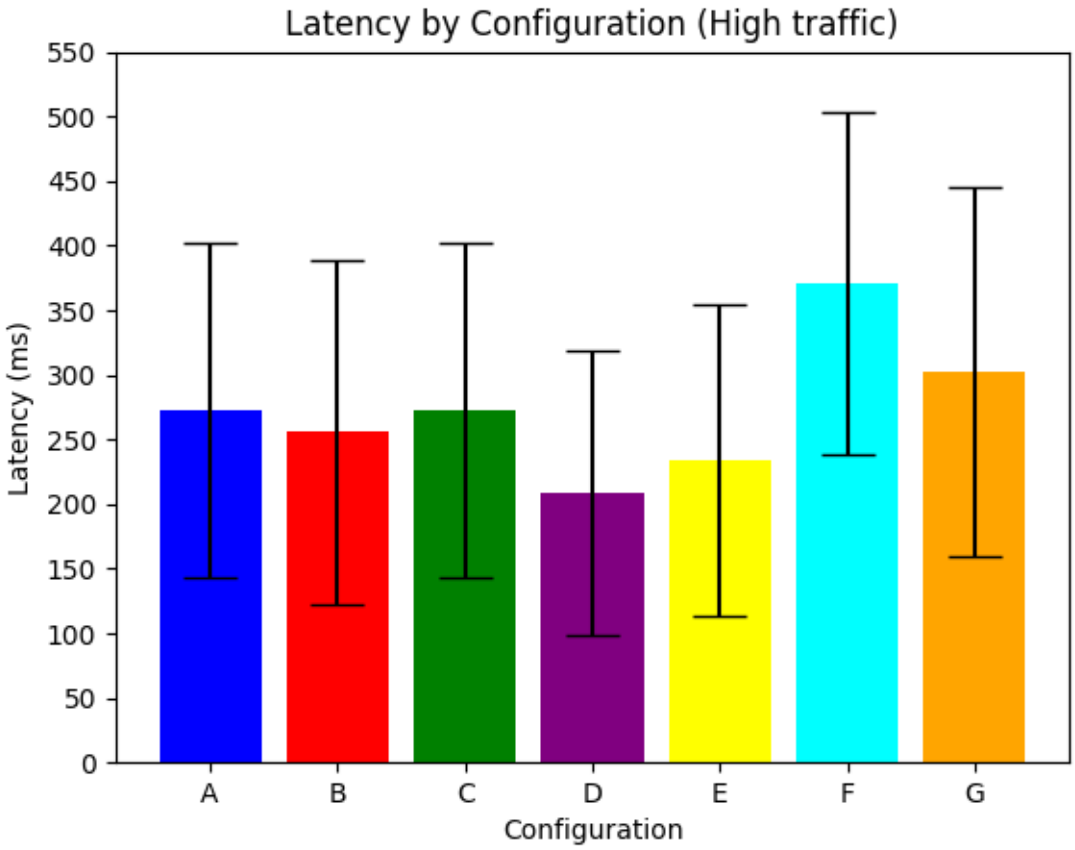


Figure 4.3: Latency (high traffic)

The latency for high traffic simulations can be seen in Figure 4.3. It can be seen that configuration D had the lowest latency figure, with latency of approximately 210 ms. Configuration F is the lowest performer with a latency figure of 370 ms. There is a 76% increase in latency between configuration D and configuration F. Similar to figure 4.2, this is likely due to the increased overhead of the 3 broker configuration with replication factor of 3 in configuration F.

It is worth noting that configuration G was the lowest performing configuration in terms of latency in the low traffic simulations. In high traffic simulations, configuration F experiences the highest latency. Comparing the low traffic simulations in Figure 4.2 and the high traffic simulations in 4.3, it can be seen that configuration F experienced an increase in latency of 123%, while configuration G only experienced an increase of 37%. The only difference in configuration between F and G is the number of consumers. Configuration F has a single consumer, while configuration G has 3 consumers. This may indicate that an increased number of consumers is better suited to higher traffic situations.

Latency by Topic

The following section shows the latency measurements of each configuration, with the latency measurements displayed on a topic by topic basis. All simulations are conducted using the architecture outlined in 3.1.

The latency measurements for low traffic simulations can be seen in Figure 4.4.

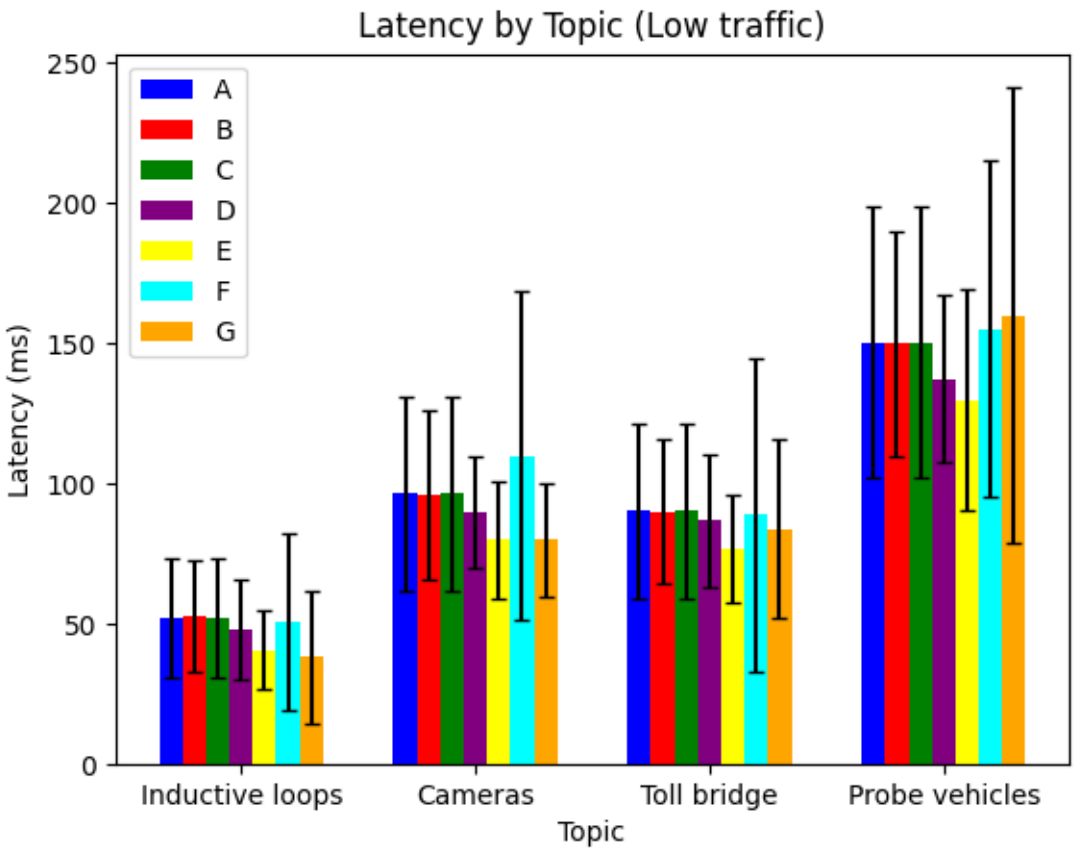


Figure 4.4: Latency by topic (low traffic)

It can be seen the inductive loops topic has the lowest latency of all the topics, with Configuration G having the lowest latency at around 35 ms. Configuration B has the highest latency of the inductive loop topics, experiencing approximately 55 ms latency. This is a 57% increase in latency.

The reason the inductive loops experience the least amount of latency is likely due to the fact that the inductive loop's data requires half the number of network hops to reach the digital twin consumer than the other topics. This is because the other topics also require the use of the enterprise server. This is outlined in 3.1.

The performance between the cameras topic and the toll bridge topic is similar in low traffic simulations. There is a 6% increase in latency with configuration E between the toll bridge topic and the cameras topic.

It is worth noting that configuration F performed significantly worse in the cameras topic than in the toll bridge topic, with the cameras topic seeing a 24% increase in latency over the toll bridge topic with configuration F. The cameras has a larger volume of messages compared to the toll bridge topic as outlined in 4.1. This could indicate that the combination of a 3 broker and single consumer setup in configuration F can't deal with high volumes of messages.

The probe vehicles topic has the highest latency measurements among the topics. Configuration E is the highest performer in both the toll bridge topic and the probe vehicles topic. Configuration E experiences a 71% increase in latency when comparing the toll bridge topic to the probe vehicles topic. This is likely due to the increased number of messages in the probe vehicles topic. The volume of messages produced per topic is outlined in table 4.1.

The latency measurements for high traffic simulations can be seen in Figure 4.5.

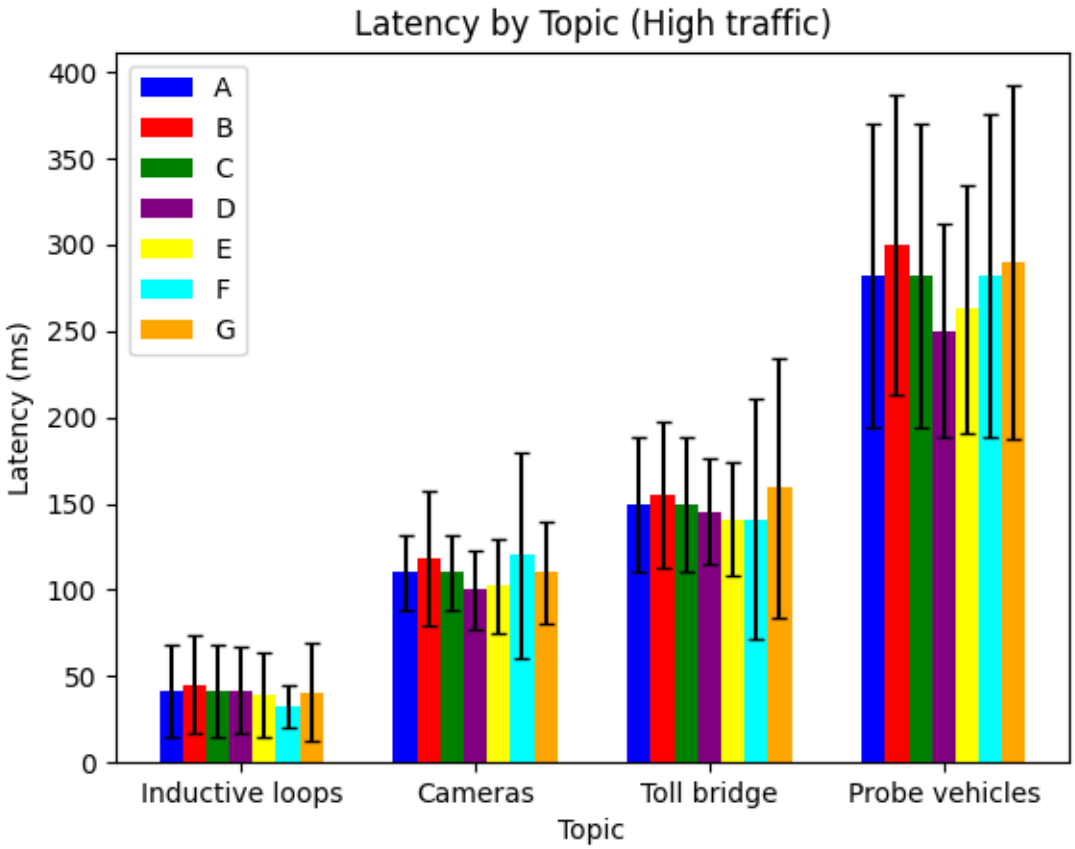


Figure 4.5: Latency by topic (high traffic)

As with low traffic simulations, inductive loops have the lowest latency in high traffic configurations. It can be seen that configuration F performs relatively better in high traffic simulations in the inductive loops topic. In the low traffic simulations, configuration F was one of the lowest performing configurations, however in high traffic simulations it is the best performing configuration for inductive loops. The error associated with configuration F is also considerably smaller in higher traffic simulations. This can be seen by comparing Figure 4.5 with Figure 4.4.

It is interesting to note the cameras topic has noticeably less latency across all configurations than the toll bridge topic. Comparing configuration D in both the cameras topic and the toll bridge topic, it can be seen that the toll bridge topic experiences an approximate 45% increase in latency compared to the cameras topic. This is unexpected behaviour as the cameras topic contains significantly more data than the toll bridge topic. In low traffic simulations 48,688 messages pass through the camera topic, while 5,691 messages pass through the toll bridge topic. The camera topic contains more partition's than the toll bridge topic, with the camera topic having 8 partitions and the toll bridge topic having 2. This may be a possible explanation for the increased latency observed in the toll bridge topic.

It can be seen that configuration F has approximately 14% increased latency compared to configuration G with regards to the cameras topic. This is contrary to the other topics, where configuration F has less latency than configuration G.

The latency of the probe vehicle topic is significantly higher than that of the other topics. Using configuration D as a comparison, the probe vehicle topic experiences approximately 160% higher latency than the cameras topic. This is likely due to the high number of probe vehicle messages being produced in comparison to the other topics. In low traffic simulations, the probe vehicles topic contains 180% more messages than the cameras topic.

It can be seen that configuration B performed poorly in all topics, with configuration B performing 9% worse than configuration A in the probe vehicle topic. Configuration B uses gzip compression while configuration A does not use compression.

Number of Consumers Comparison

The effects of the number of consumers can be seen directly in Figure 4.6, which compares the average latency of configuration B and configuration D.

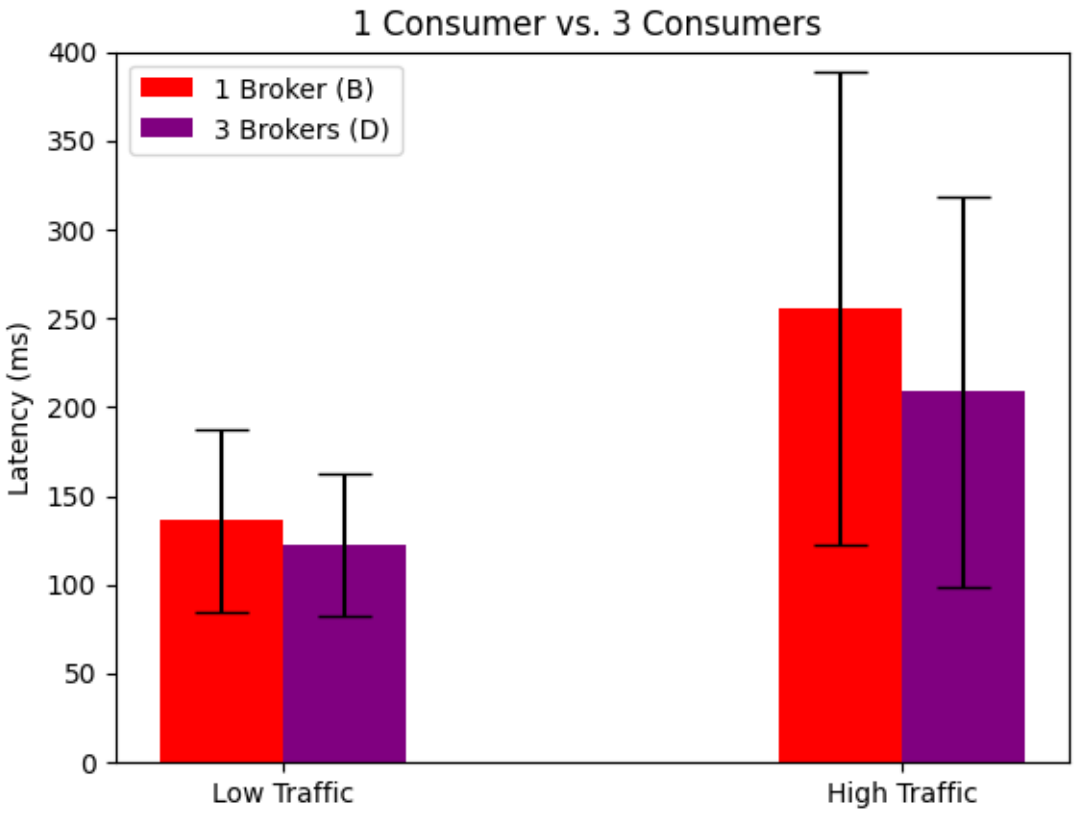


Figure 4.6: Comparison of 1 consumer versus 3 consumers

Configuration B and configuration D have identical configurations except the number of consumers is increased from 1 in configuration B to 3 in configuration D. It can be seen that for both low and high traffic simulations configuration D experienced lower latency than configuration B. Configuration D experiences 11% less latency in low traffic simulations, and 23% less latency in high traffic simulations. This indicates that a multiple consumer setup is favourable over a single consumer setup providing the total number of consumed messages remains the same.

Broker Hardware Comparison

A similar comparison is conducted between the hardware in which the Kafka broker runs. Configurations B and E are compared, with configuration B running on a LINUX/UNIX t2.large, and configuration E running on a LINUX/UNIX t2.xlarge. The difference between these instances are outlined in section 4.2.

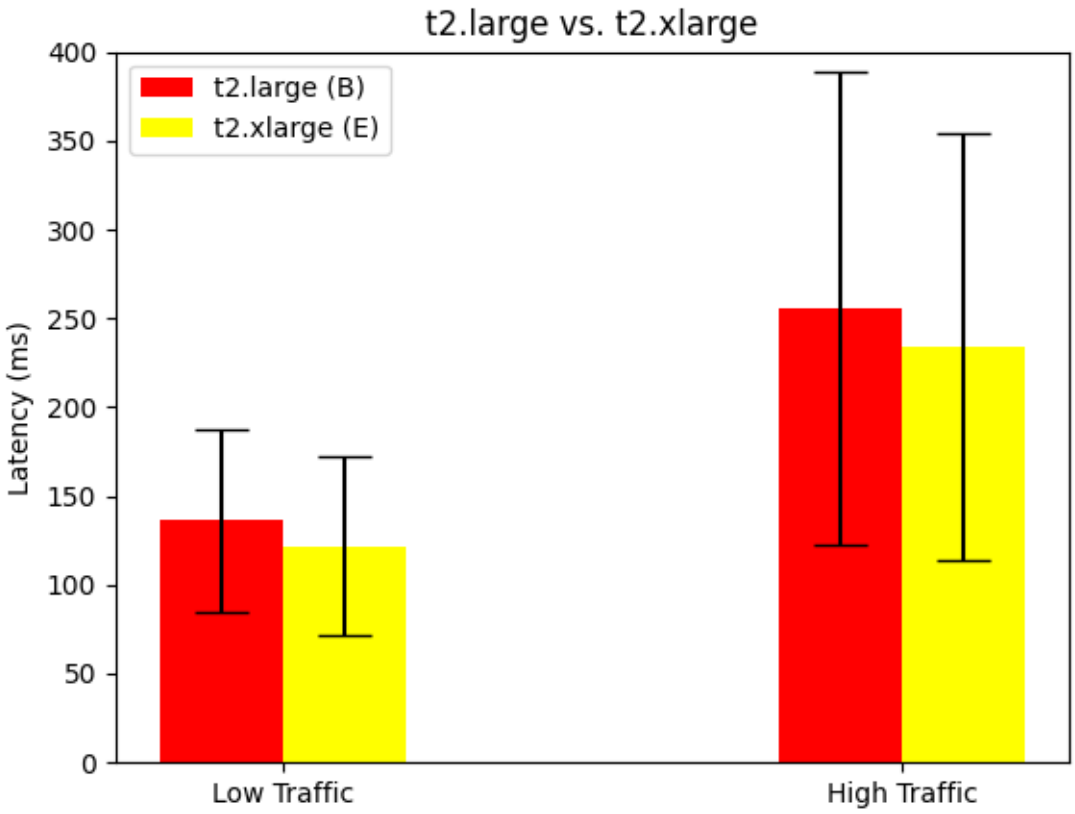


Figure 4.7: Kafka broker hardware comparison

It can be seen in Figure 4.7 that configuration E has lower average latency than configuration B for both high and low traffic situations. In low traffic simulations configuration E experiences 12% less latency than configuration B. In high traffic simulations configuration E experiences 11% less latency than configuration B. This implies that if all other variables are fixed, a more powerful machine will provide lower latency measurements.

Number of Brokers Comparison

Figure 4.8 shows the effects of increasing the number of brokers from 1 to 3 brokers. All other variables are fixed in this comparison. Configuration B has 1 Kafka broker and configuration F has 3 Kafka brokers, with a replication factor of 3.

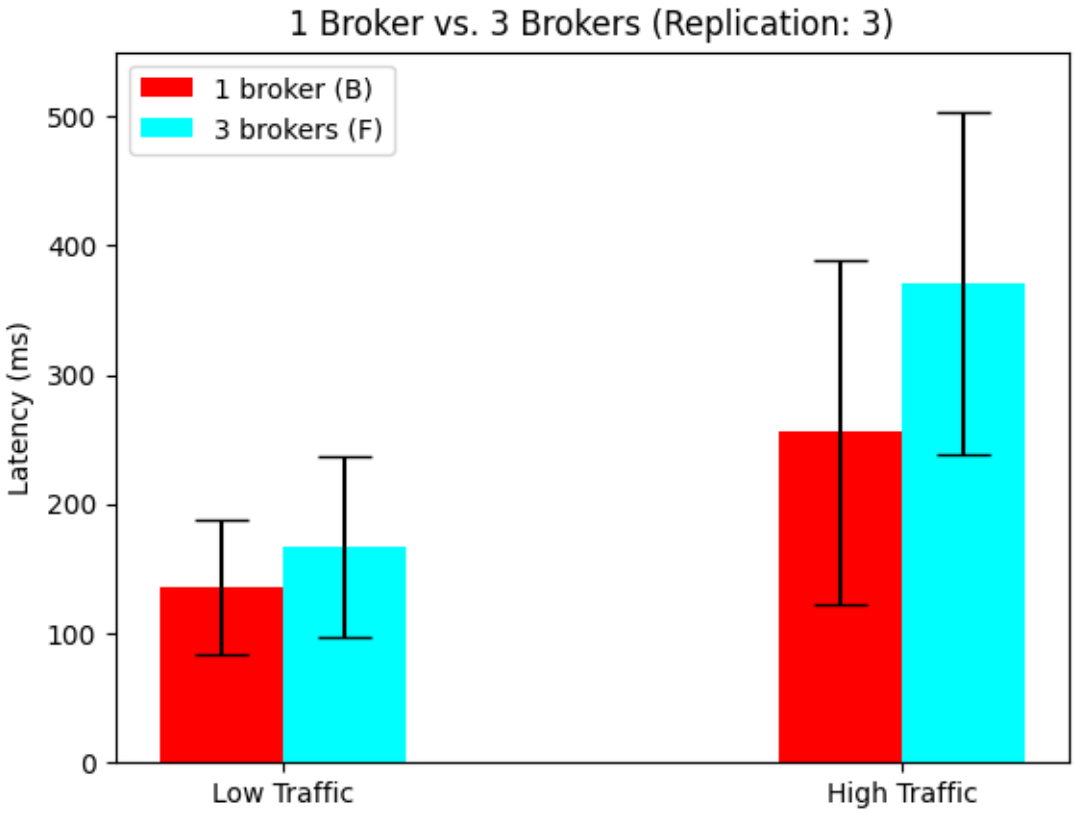


Figure 4.8: Number of Kafka brokers comparison

It can be seen that in low traffic simulations, configuration B is approximately 21% faster than configuration F. In high traffic situations configuration B is approximately 44% faster than configuration F. Clearly a single broker provides lower latency than 3 brokers in simulation scenarios. The trade off here is that a single broker has a single point of failure. Multiple brokers improves the availability of the service. A replication factor of 3 ensures that the messages are replicated across all 3 brokers. It would be desirable to maintain this highly available service while lowering the latency.

Compression Comparison

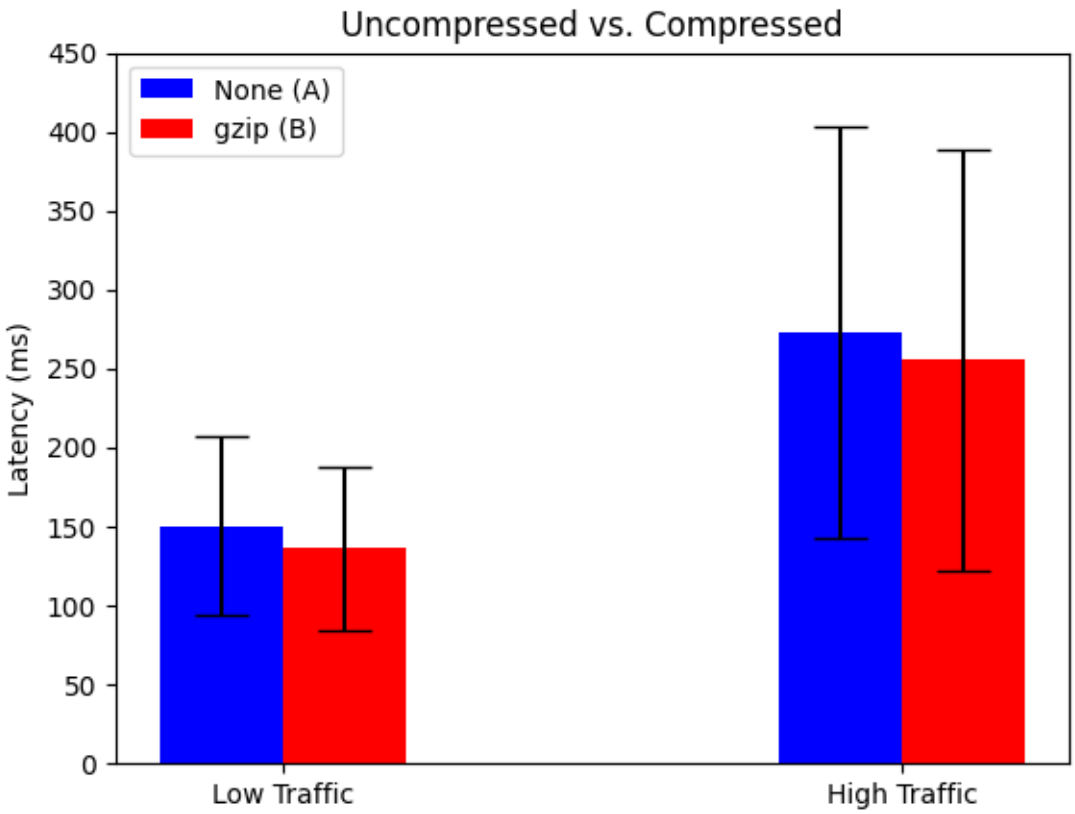


Figure 4.9: Kafka broker compression comparison

The effects of compression on the system can be seen in Figure 4.9. It is clear that configuration B which employs gzip compression, has lower latency than configuration A. In low traffic simulations gzip compression experiences 7% less latency than no compression. In high traffic simulations this figure rises to 8%. This is contrary to the results in Figure 4.5, which saw configuration B perform slightly worse than configuration A in high traffic scenarios.

4.5.2 Benchmark Results

Each configuration is benchmarked as outlined in section 4.3.2. The benchmark tests do not incorporate the enterprise server. All data is streamed directly to the digital twin Kafka cluster. There are 4 topics, each with 250,000 messages, that are streamed through each Kafka configuration to measure the throughput. The throughput of each configuration can be seen in Figure 4.10.

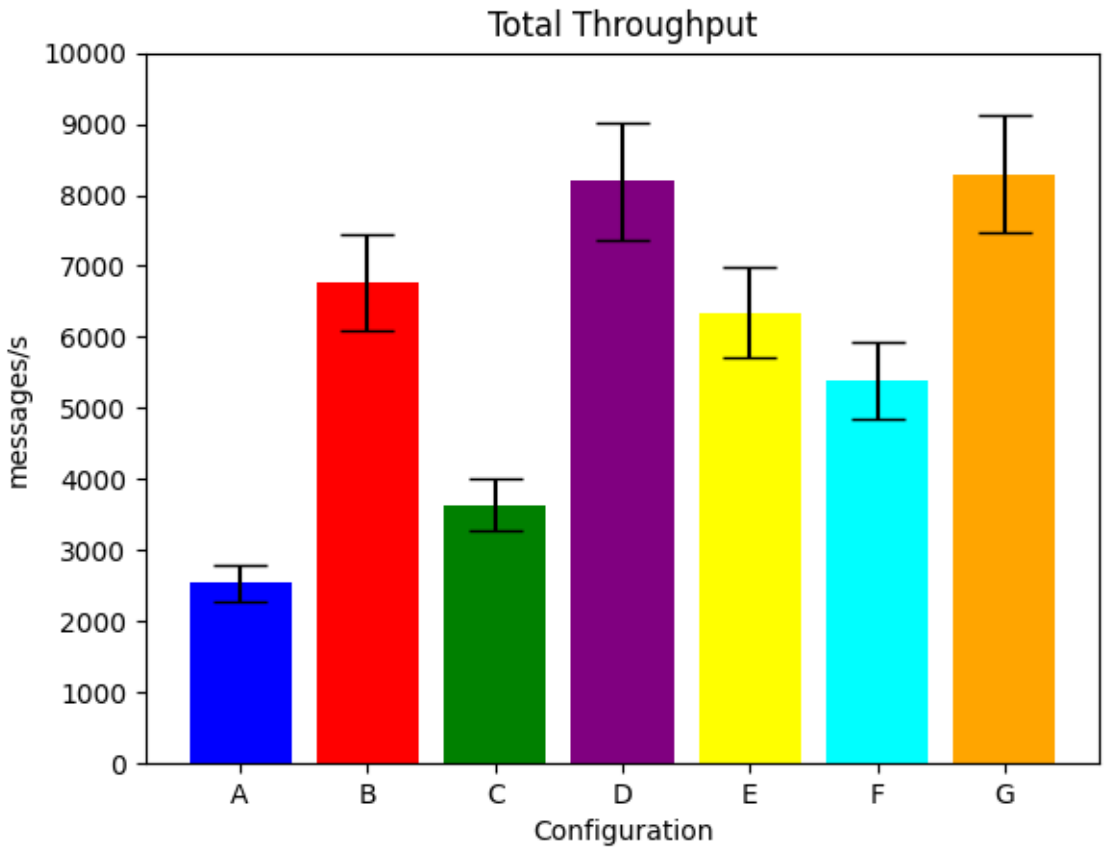


Figure 4.10: Benchmark throughput by configuration

Upon examining Figure 4.10, it is clear that configuration A has the lowest throughput performance of all configurations, with 2500 messages per second passing through Kafka in this configuration. The highest performing configurations are configuration D and G, with both configurations getting around 8100 messages per second. These configurations have 3 consumers while all other configurations have a single consumer.

Configurations A and C have identical configurations bar the fact configuration C used gzip compression while configuration A does not use compression. It can be seen by comparing configuration A to C that compression increases the throughput of the system by approximately 40%.

It can be seen that the broker hardware had no significant performance benefits in the throughput benchmark tests. Configuration B and configuration E have identical configurations, however configuration B is using a t2.large and configuration E is using a t2.xlarge. It can be seen that configuration B actually experienced 6% higher throughput than configuration E, despite running on less powerful hardware.

Configuration B has 160% higher throughput than configuration A. The only difference between these configuration is that configuration B uses gzip compression while configuration A has no compression. It is clear in this test setup that compression allows for higher throughput than no compression.

Comparing configuration A and configuration C, it can be seen that a throughput increase of 43% is observed. It could therefore be said that increasing the batch size increases the throughput of the system as this is the only notable difference between configurations A and C. However it could be argued that increasing the batch size too much could increase the latency of the system, as some messages will not be sent to the broker as soon as they are made available by the producer. This may lead to increased latency for some of the earlier messages in the batch.

The throughput figures for each configuration can be seen in table 4.5:

	A	B	C	D	E	F	G
messages/s	2538	6768	3637	8193	6347	5379	8294

Table 4.5: messages/s by configuration

4.5.3 Summary of Results

There are a number of key take-ways from the tests conducted. The knowledge gained from these tests is used to design the final solution in section 4.11.

- It is clear from Figure 4.8 that a single broker is capable of providing lower latency measurements than a multiple broker setup, however single broker setups do not have a replication factor above 1, so there is less overhead compared to a multi-broker configuration.
- The availability of the service is greatly improved with multiple brokers. A single broker means that if that broker were to fail, the service stops. If the design has multiple brokers then a single broker failure will not halt the message delivery service.
- Assuming the number of total consumed messages is fixed, multiple consumers provide a considerable performance benefit over a single consumer. This is evident in Figure 4.6.
- It seems reasonable to assume that more powerful hardware will lead to a better service. This is backed up by the results in 4.7. However no improvements in performance can be seen in 4.5.
- Compression offers considerable performance benefits. It can be seen in 4.9 that utilising compression in the design leads to greater performance in terms of latency and throughput.
- Although batching can possibly increase the throughput of the system, Figure 4.2 and Figure 4.3 show that increased batch size does not offer any significant performance increase in terms of latency over the default batch size. It can also be seen in Figure 4.10 that compression yields far better throughput results than increased batching.

4.5.4 Final Solution

The final solution is designed based on the findings outlined in 4.5.3. The final design will utilise a 3 broker configuration. This is due to the increased availability offered by multiple brokers. These brokers will utilise t2.xlarge instances in order to provide the best performance possible. It is clear that multiple consumers offers better performance. The final design will incorporate 1 consumer per topic. The producers in the final design will use gzip compression. The benefits of compression is clear in 4.9. The batching size is set to the default value due to reasons outlined in 4.5.3.

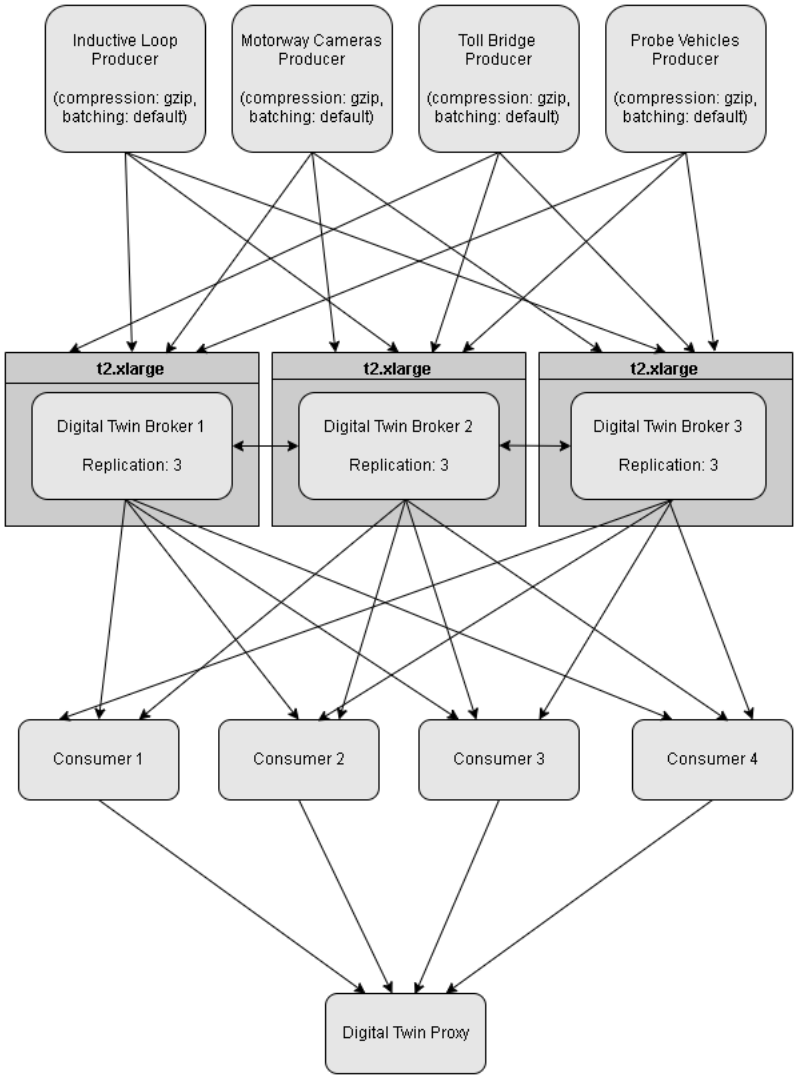


Figure 4.11: Final Kafka broker configuration

4.5.5 Performance of Final Solution

The performance of the final solution is compared to that of the configurations outlined in 4.4. Both simulation tests and benchmark tests are performed on the final solution similar to the tests outlined in 4.3.1 and 4.3.2.

Simulation Results

Comparing the final solution to the configurations under test outlined in 4.4, it can be seen in Figure 4.12 that the final solution performs better than configuration G in low traffic simulations, with configuration G having around 8% higher latency than the final solution. It can also be seen that configuration F has 24% less latency than the final solution.

It is clear that the final solution experiences more latency than the single broker configurations, with the final solution experiencing 70% more latency than configuration E. However a single broker solution does not offer a highly available service as there is a single point of failure. If the broker on configurations A through E were to fail, the service would stop. This is not the case in the 3 broker configurations were the remaining 2 brokers can continue the service.

Replication also minimises the chance of data loss, as the only situation in which a message can be lost is if the broker which has received that message immediately fails after acknowledging the message, but prior to replicating the message. For these reasons a 3 broker configuration is favourable over a single broker configuration despite the added latency cost.

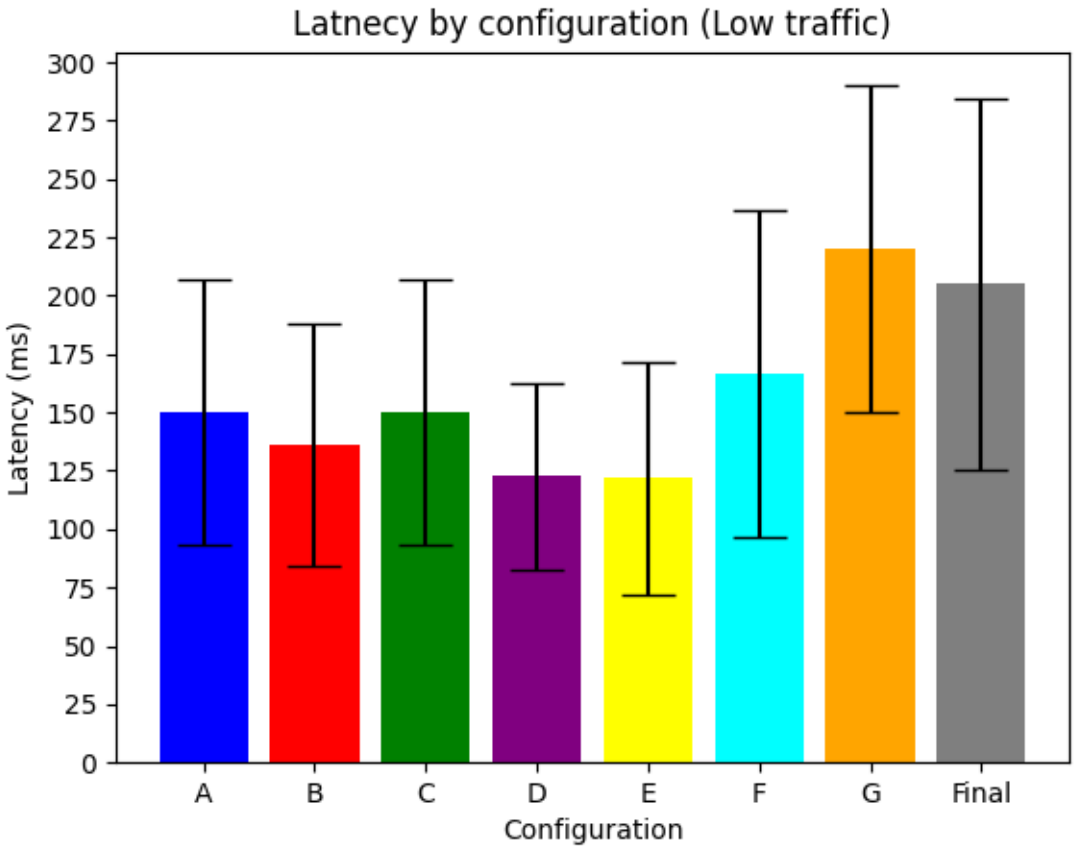


Figure 4.12: Final solution latency comparison (low traffic)

In high traffic simulations, it is clear that the final configuration is the highest performing 3 broker configuration. It can be seen that configuration F has 35% higher latency than the final solution. This is likely due to the increased number of consumers found in the final solution compared to configuration F, which has a single consumer.

The final solution's brokers run on more powerful machines than in configuration F. This may also be a contributing factor to the increased performance in the final solution.

Comparing configuration G and the final configuration, it can be seen that configuration G experiences 9% higher latency than the final design. This in likely due to the extra broker in the final design, as well as the more powerful t2.xlarge brokers used in the final solution compared to the t2.large machines used in configuration F. It can also been seen than the final design has a lower interquartile range than configuration G. This suggests that the final design's latency measurements have less spread than that of configuration G. This is preferable as it allows for higher consistency in latency predictions.

In high traffic simulations the final design offers similar performance to configuration A, configuration B, and configuration C despite these configurations being single broker systems which offer considerably less availability.

It is interesting to note that configuration D experienced 71% less latency than the final solution in low traffic simulations, but only 31% less latency in higher traffic simulations. This suggests that the final solution is less susceptible to performance decline with increased traffic compared to a single broker configuration such as configuration D.

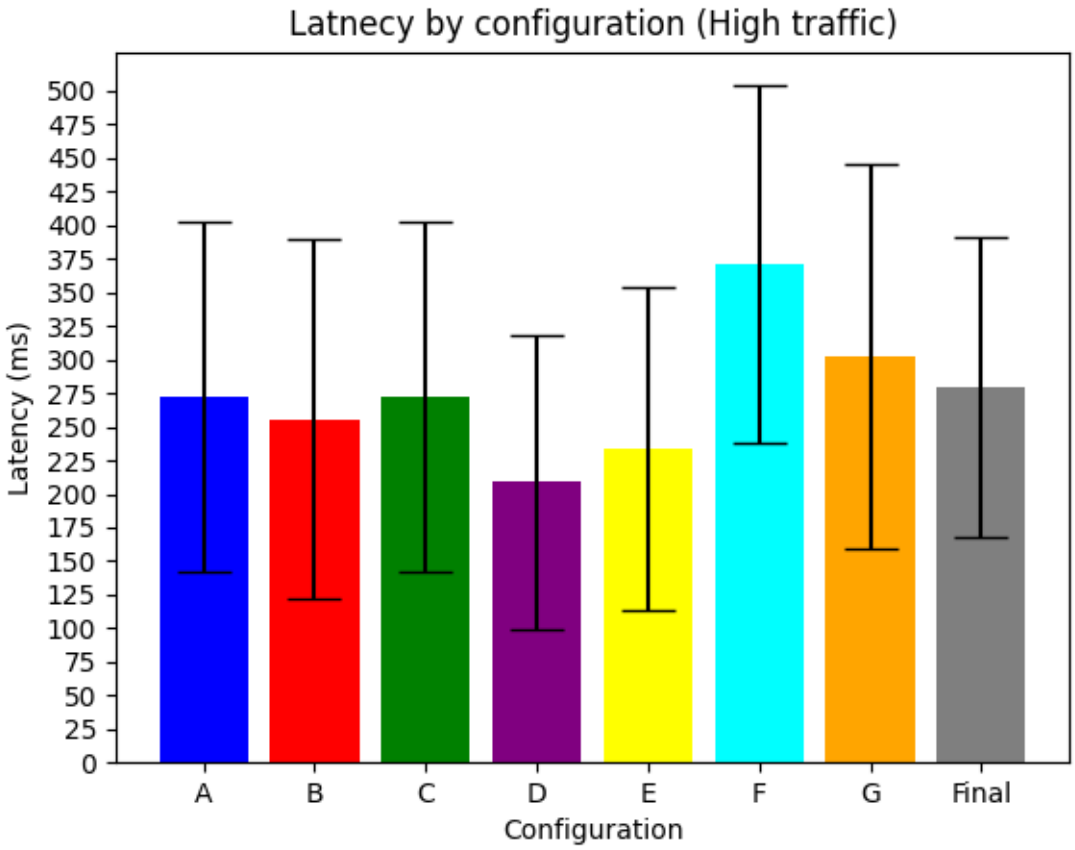


Figure 4.13: Final solution latency comparison (high traffic)

The final design outlined in 4.11 is benchmarked as outlined in section 4.3.2. The results can be seen in Figure 4.14.

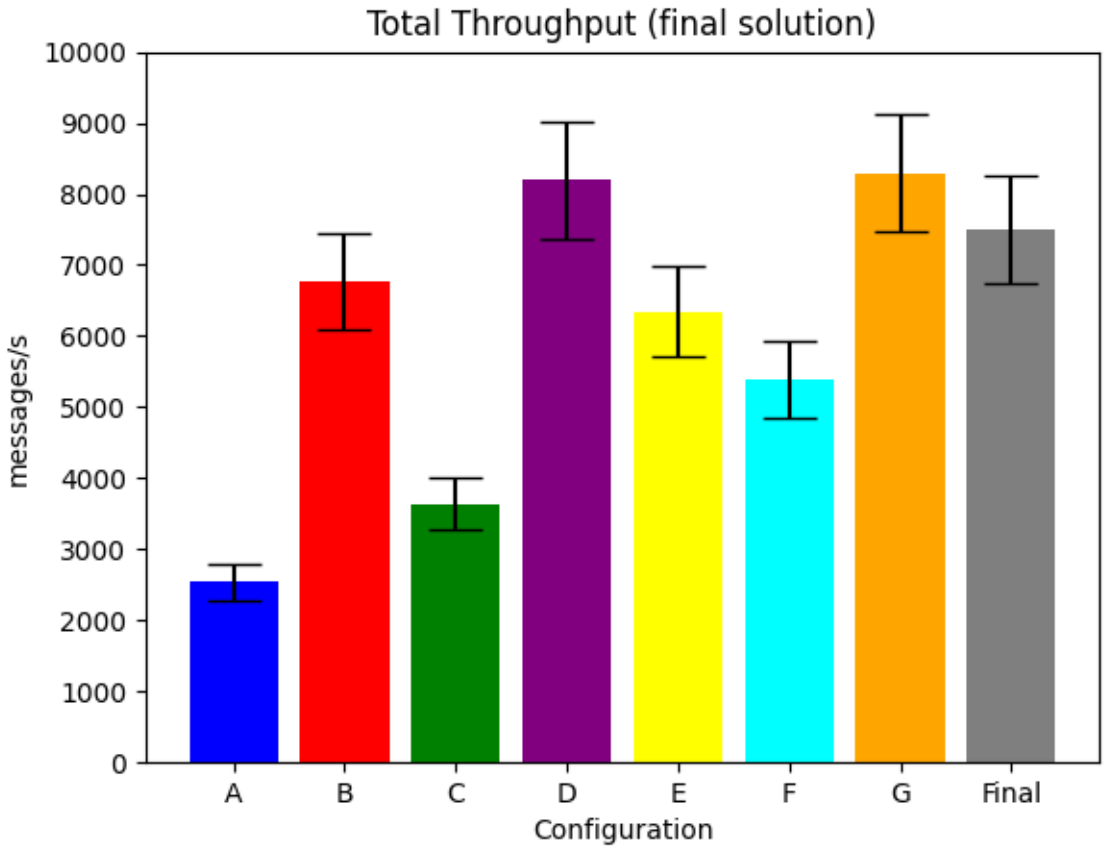


Figure 4.14: Throughput of final solution

It can be seen that the final design actually performed marginally worse than configuration D and configuration G, with the final solution providing 8% less throughput than configuration D, and 9% less throughput than configuration G. The reasons for this are unclear.

Effects of Distance to Kafka Brokers

The following section compares the performance of the final solution outlined in Figure 4.12 in 2 locations. These locations are chosen due to their relative distance to the EC2 instances in which Kafka is hosted. The objective is to measure the affects of the distance between the producers/consumers and Kafka. This is important as it can aid in determining the best broker placement for optimum performance.

The high and low traffic simulation tests which are outlined in 4.3.1 are conducted in each of these locations.

Location 1 is 25 km from the EC2 instances in which Kafka is hosted (location 1 is were all previous tests were conducted). Location 2 is 2.5 km from the EC2 instances in which Kafka is hosted.

The latency measurements for low traffic simulations conducted using the final solution in both location 1 and location 2 can be seen in Figure 4.15.

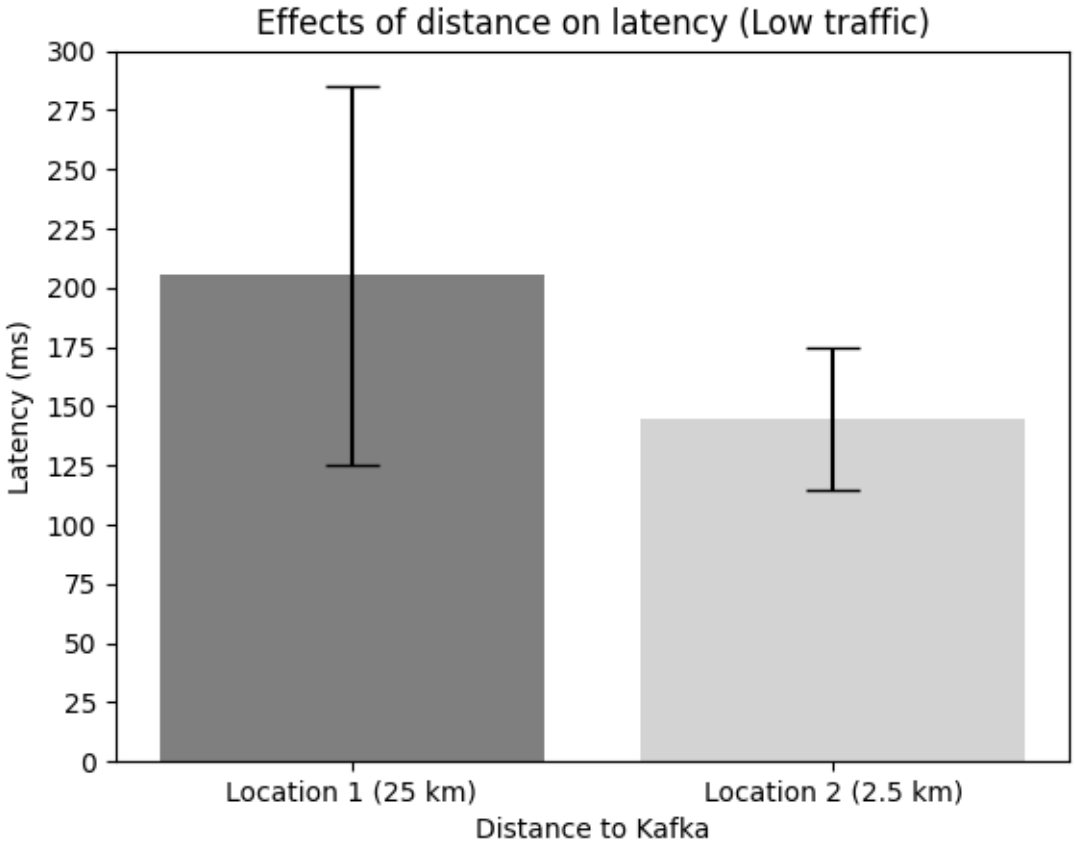


Figure 4.15: Effect of distance on latency (low traffic)

It can be seen in Figure 4.15 that location 1 experienced 46% higher latency than location 2. The IQR of location 2 is also significantly lower than location 1, indicating location 2 has more consistent latency measurements.

The latency measurements for high traffic simulations conducted using the final solution in both location 1 and location 2 can be seen in Figure 4.16.

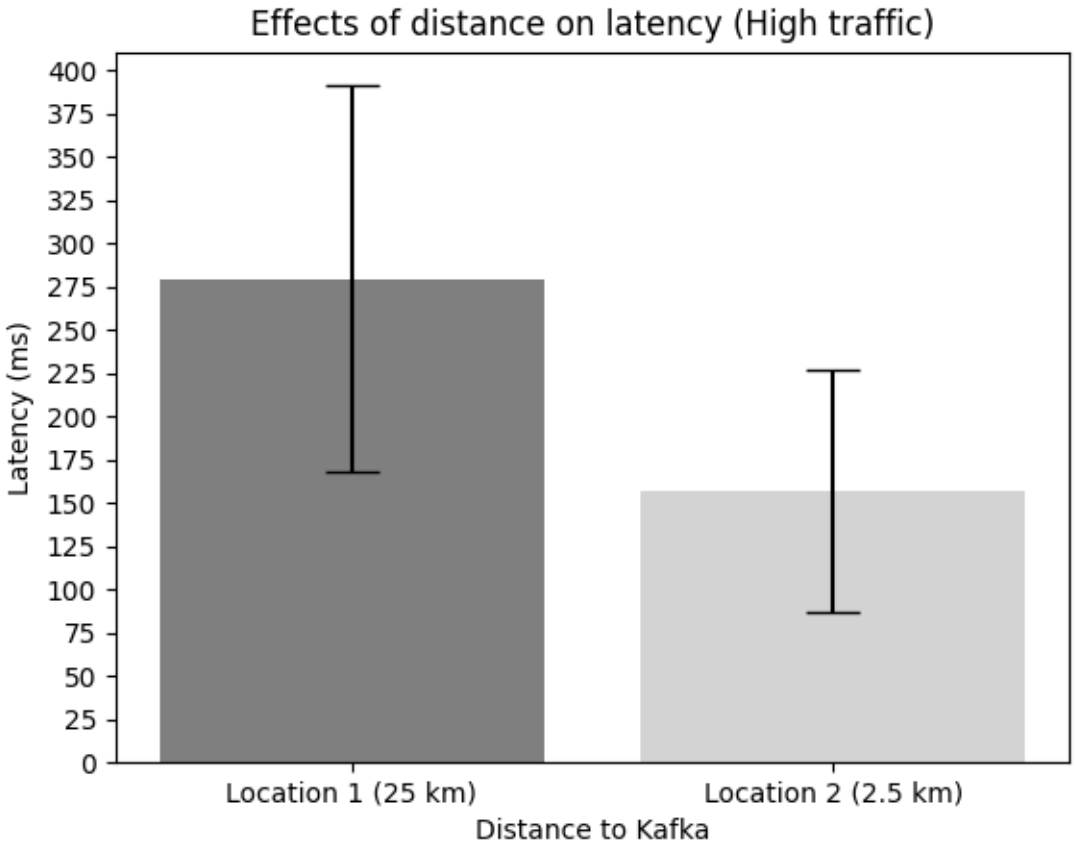


Figure 4.16: Effect of distance on latency (high traffic)

It can be seen in Figure 4.16 that location 1 experienced 72% higher latency than location 2. This is a 26% improvement on the latency difference between location 1 and location 2 in Figure 4.15. This suggests that with increased traffic, the distance between the producers/consumers has an increased effect on the latency observed by the system. Since there are only 2 locations under test in this report, these results are far from conclusive. Further testing in various locations would be required to draw a meaningful relationship between the latency of the system and the distance to Kafka. Despite this, it stands to reason that the closer the producers/consumers are to Kafka, the less latency the system will experience.

Throughput tests similar to that of those in Figure 4.10 are conducted on the final solution in location 1 and location 2. The results can be seen in Figure 4.17.

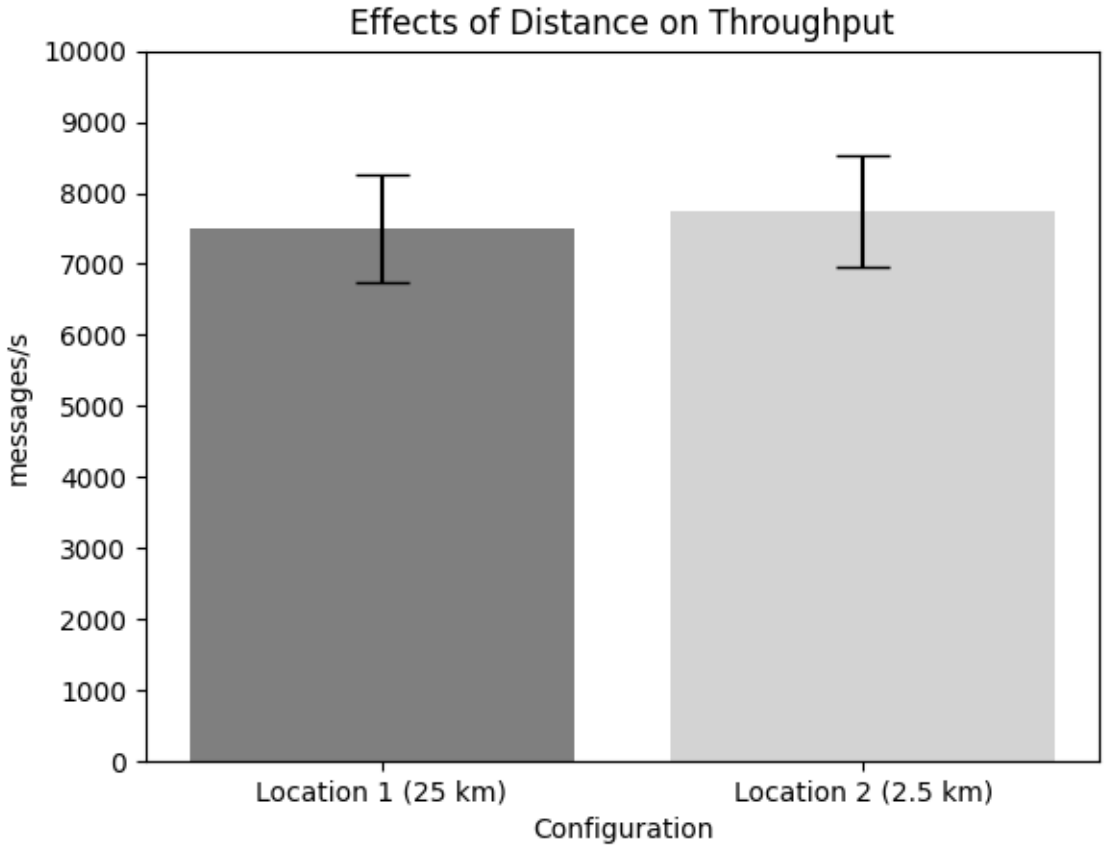


Figure 4.17: Effect of distance on throughput

It can be seen in Figure 4.17 that the difference in throughput between location 1 and location 2 is marginal. Location 2 allowed for 2.6% more throughput, however this difference is small and is well within the margin of error for this experiment. It seems likely that the distance between the producers/consumers has little to no effect on the total throughput the system, however further experiments in varying locations are needed to confirm this.

5 Summary

This project focuses on developing a communication architecture that is suitable for transportation digital twins. This requires a system that is highly available, has low latency, and high throughput.

The goal of the project is to design a communication method between motorway sensors and a digital twin. Several popular middleware technologies are considered in this research. Apache Kafka is selected as the middleware technology used in the design due to its high throughput capabilities, low latency, and its use of replication, which allows for a highly available service.

The design has several components. Apache Kafka producers are used to stream data from the traffic sensors to the digital twin Kafka cluster. It is assumed that certain sensor data will not be directly available to the producers. This data will be obtained through 3rd party enterprise servers which will produce data to the digital twin Kafka cluster. The data from the digital twin Kafka cluster is consumed by the digital twin consumer(s). The data ingested from these consumers is used to feed the digital twin.

The design is evaluated using simulated motorway sensors. The common source in which the simulated sensors draw their data is a SUMO simulation of the M50 motorway. The performance of several Kafka configurations are assessed using tests which evaluate the latency and throughput of the system.

The final design is a 3 broker configuration with replication across all brokers. This allows for a highly available service. The design uses gzip compression in order to reduce the latency. The Kafka brokers are deployed on AWS EC2 t2.xlarge instances. A consumer per topic system is employed in the design to improve the latency and throughput of the design in high traffic situations.

The design is tested in various location in order to determine the ideal Kafka broker placement for optimum performance. It was found that systems which are closer to the producers/consumers have lower latency. The effects of broker placement on throughput are minimal.

6 Conclusion

From the evaluation of the final design, it is clear that Apache Kafka is capable of providing a service which fits the requirements of the system.

It was found that a single broker cluster experiences less latency than a multiple broker cluster, however single broker clusters have a single point of failure and do not make for a highly available service. It was found in figure 4.9 and 4.7 that a combination of compression and increased broker computing power can decrease the latency by approximately 17%. It is clear from the findings in 4.6 that multiple consumers offer considerable performance benefits when compared to a single consumer.

The final design is 3 broker setup with replication across all brokers. This makes the design highly available, providing minimal loss of service. The design used gzip compression and powerful broker hardware, as well as a consumer per topic system, in which each topic has a dedicated consumer. These design choices aid in reducing the latency of the design. This is supported by findings in figure 4.12 and 4.13. The multiple consumer setup allows the design to handle high throughput. This is supported by findings in figure 4.14.

It is clear that Kafka cluster placement is a critical design choice if latency is to be kept to a minimum. In low traffic simulation scenarios, with the Kafka cluster situated 2.5 km from the producers/consumers, the final design is capable of streaming data from producer to consumer in approximately 145 ms. In high traffic scenarios, this figure rises 7% to 156 ms. If the relative producer/consumer to cluster distance is raised to 25 km, these figures rise 46% to 205 ms, and 72% to 280 ms respectively. The total throughput of the system is approximately 7600 messages per second regardless of cluster placement.

This research provides valuable insight into the use of Apache Kafka as the communication method between motorway sensors and a digital twin. The project designed a communication framework based on the results of experimental research conducted in the form of motorway traffic simulations using virtualised traffic sensors.

The results of these experiments aided in designing a highly available system which is capable of providing high throughput while maintaining low latency. The research also provided insight into the ideal cluster location, suggesting that clusters located closer to the producers and consumers will lower the latency of the system, however it will not necessarily increase the maximum throughput of the system.

6.1 Future Work

There are several research topics which can be explored following this project. The immediate future work concerning this project should be implementing a digital twin in which the architecture outlined in this dissertation is the communication method between the sensors and the digital twin. This would require unpacking the data at the consumer side of the system and attempting to recreate the motorway simulation from the sensor data.

There are also several avenues to explore with regards to improving the evaluation and design of the framework. There are several limitations to the simulation which are not addressed in this research, mainly that the producer scripts run sequentially within the simulation. Further research would run these producers in parallel to faithfully represent the real-world sensors, which of course can produce data at the same time. This may be possible by controlling the motorway simulation from multiple TraCI clients which are each running producer scripts [German Aerospace Center, 2022d].

In this research, all the producers are hosted on a single client. In reality, producers will be in various locations, with each location likely being different distances from the Kafka cluster. Further research could implement multiple clients in various locations to more accurately represent the producers. Implementing multiple clients in different locations with producers running in parallel would make for a much more faithful representation of real life.

It is clear from the research that multiple consumers performs better than a single consumer, however it is unclear whether this is due to multiple consumers consuming in parallel, allowing for messages to be consumed simultaneously, or if this is due to the polling frequency of the consumer, with multiple consumers simply being able to poll faster as each consumer in a multiple consumer setup is consuming less messages than the consumer in the single consumer setup. Further research may provide answers as to what degree the polling frequency of consumers limits the number of messages that can be consumed. Further optimisations may be possible to allow consumers to poll at a higher frequency while still making the data available to the digital twin.

Further experimentation with various Kafka cluster locations could provide further insight into the relationship between the latency of the system and the distance to Kafka. Although this project did experiment with different cluster locations, this research was limited and there is room for further research. An interesting research topic would be to experiment with various Kafka cluster locations and hypothesis an idealised location relative to the M50 motorway which would provide the lowest latency possible for the digital twin.

In this design, the data communicated between Kafka and the producers and consumers is sent in plain text. Apache Kafka is capable of utilising security protocols such as SSL, however this is not explored in this project. Further work may include on-boarding suitable security measures in order to improve the security of the design.

Bibliography

J. Kreps, N. Narkhede, and J. Rao. Kafka: A distributed messaging system for log processing. Proc. Int. Workshop Netw. Meets Databases, 2011.

Dobbelaere, Philippe, and Kyumars Sheykh Esmaili. Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper. pages 227–238, 06 2017. doi: 10.1145/3093742.3093908.

Fu. Guo, Zhang. Yanfeng, and Yu. Ge. A fair comparison of message queuing systems. *IEEE Access*, 9:421–432, 2021. doi: 10.1109/ACCESS.2020.3046503.

Vinoth Chandar Alok Nikhil. Benchmarking apache kafka, apache pulsar, and rabbitmq: Which is the fastest?, August 2020. URL `https://www.confluent.io/blog/kafka-fastest-messaging-system/`.

Garmin. Gps 16x technical specifications, September 2008. URL `https://docs.rs-online.com/9837/0900766b80e05373.pdf`.

Electro Automation. Sensys traffics sss (speed safety system), 2021a. URL `https://www.electroautomation.com/page/downloads`.

Medium. How computer vision can change the automotive industry, 2018. URL `https://miro.medium.com/max/720/1*qmnZgXVuIlx9rreFjeO0sg.jpeg`.

Dublin City Council. Dublin city council traffic cameras, 2017. URL `https://www.dublincity.ie/dublintraffic/`.

C. Sowers. The future of traffic monitoring – a new perspective using drones. *Conférence ACPSER, Toronto*, June 2017. URL `https://www.unb.ca/research/transportation-group/_resources/pdf/research-papers/traffic-monitoring-future.pdf`.

TII. Traffic counts for transport infrastructure tii, 2022. URL `https://trafficdata.tii.ie/publicmultinodemap.asp`.

IBM Digital Twin Exchange. *What is a digital twin?* IBM, 2021.

Central Statistics Office. *Transport Omnibus 2019 - Road Traffic Volumes*. Central Statistics Office, 2019.

Simon Carswell. *M50 blues: Ireland's busiest road, Dublin's biggest car park*. The Irish Times, 2017.

IBM. *Publish/subscribe overview*. IBM, 2021.

Hadoop. Hdfs architecture guide, 2008. URL https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

Amazon Web Services. What is amazon s3?, 2022a. URL https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

Sunny Garg. What makes apache kafka so fast?, 2019. URL https://medium.com/@sunny_81705/what-makes-apache-kafka-so-fast-71b477dcbf0.

Colin McCabe. Apache kafka needs no keeper: Removing the apache zookeeper dependency, May 2020. URL https://www.confluent.io/blog/removing-zookeeper-dependency-in-kafka/.

Praveen Kumar Donta, Satish Narayana Srirama, Tarachand Amgoth, and Chandra Sekhara Rao Annavarapu. Survey on recent advances in iot application layer protocols and machine learning scope for research directions. *Digital Communications and Networks*, 2021. ISSN 2352-8648. doi: https://doi.org/10.1016/j.dcan.2021.10.004. URL https://www.sciencedirect.com/science/article/pii/S2352864821000845.

Flavio P. Junqueira, Ivan Kelly, and Benjamin Reed. Durability with bookkeeper. *SIGOPS Oper. Syst. Rev.*, 47(1):9–15, jan 2013. ISSN 0163-5980. doi: 10.1145/2433140.2433144. URL https://doi.org/10.1145/2433140.2433144.

Jiang Yongguo, Liu Qiang, Qin Changshuai, Su Jian, and Liu Qianqian. Message-oriented middleware: A review. *2019 5th International Conference on Big Data Computing and Communications (BIGCOM)*, pages 88–97, 2019.

Garmin. What is gps?, 2021a. URL https://www.garmin.com/en-US/aboutgps//.

US Government. Gps accuracy, 2021. URL https://www.gps.gov/systems/gps/performance/accuracy/.

NVS. Nv08c rtk gnss device, 2021. URL http://www.nvs-gnss.com/products/receivers/nv08c-rtk-m.html.

TomTom. Real time historical traffic, 2012. URL https://download.tomtom.com/open/crm/lib/docs/licensing/RTTHT.EN.pdf.

Garmin. Garmin traffic data, 2021b. URL https://discover.garmin.com/en-IE/traffic/.

Google. Google location services, 2021a. URL https://policies.google.com/technologies/location-data?hl=en-US.

Google. Find improve your location's accuracy - android - google maps help, 2021b. URL https://support.google.com/maps/answer/2839911?hl=en&co=GENIE.Platform%3DAndroid.

Google. Get started with google assistant driving mode, 2021c. URL https://support.google.com/assistant/answer/10217503?hl=en.

P. Linnartz, J. Ivhs: Probe vehicles, 2010. URL http://www.wirelesscommunication.nl/reference/chaptr01/roadtrin/ivhsprob.html.

Government of Ireland. Gtfs real time for dublin bus, bus eireann, and go ahead services, August 2020. URL https://data.gov.ie/blog/gtfs-r-real-time-for-dublin-bus-bus-eireann-and-go-ahead-services.

DHL Paket GmbH. Live tracking, 2012. URL https://www.dhl.de/en/privatkunden/pakete-empfangen/sendungen-verfolgen/live-tracking.html.

Cambridge Pixel. Radar terminology, 2019. URL https://www.cambridgepixel.com/support/radar-terminology/.

Nanoradar. Single lane vehicle over speed detection radar, 2020. URL https://www.nanoradar.com.cn/Traffic-radar/single-lane-vehicle-over-speed-detection-radar-tsr10?gclid=CjwKCAjw2vOLBhBPEiwAjEeK9hRL_NQHBpHUL9O_AeXLaKJ5He0SHK_8nkm94BuvRhME2BTqgYg58BoC0r0QAvD_BwE.

Sensy Networks. Sensy networks snaps profession 2.20, 2021. URL https://snsys.my.salesforce.com/sfc/p/#700000008X1I/a/70000000PdPS/JzH1egvJTohSao_ql9qVy0sh0MshwDBrF8xDQido65E.

Electro Automation. 3m anpr cameras, an integrated anpr camera, 2021b. URL https://www.electroautomation.com/page/downloads.

C. Gibbs. Using traffic camera images to derive an indicator of busyness: experimental research. September 2020. URL https://www.ons.gov.uk/economy/economicoutputandproductivity/output/methodologies/usingtrafficcameraimagestoderiveanindicatorofbusynessexperimentalresearch.

Liang H. Li H. Song, H. Vision-based vehicle detection and counting system using deep learning in highway scenes. 2019. doi: https://doi.org/10.1186/s12544-019-0390-4.

Saleh Javadi, Mattias Dahl, and Mats I. Pettersson. Vehicle speed measurement model for video-based systems. *Computers Electrical Engineering*, 76:238–248, 2019. ISSN 0045-7906. doi: https://doi.org/10.1016/j.compeleceng.2019.04.001. URL https://www.sciencedirect.com/science/article/pii/S0045790618317774.

Sanderson Associates. How are vehicles detected at traffic signals?, 2021. URL https://www.traffic-signal-design.com/vehicles-detected.htm.

Hughes Trueman Paul Briedis. The accuracy of inductive loop detectors. *24th ARRB Conference – Building on 50 years of road and transport research, Melbourne, Australia 2010*, 2010. URL https://kipdf.com/the-accuracy-of-inductive-loop-detectors_5ae74e407f8b9a09528b467a.html.

German Aerospace Center. Sumo user documentation, 2022a. URL https://sumo.dlr.de/docs/index.htmll.

German Aerospace Center. nterfacing traci from python, 2022b. URL https://sumo.dlr.de/docs/TraCI/Interfacing_TraCI_from_Python.html.

Dasgupta, Rahman, Lidbe, Lu, and Jones. *A Transportation Digital-Twin Approach for Adaptive Traffic Control Systems*. Transportation Research Record, 2021.

Sathish A. P. Kumar, R. Madhumathi, Pethuru Raj Chelliah, Lei Tao, and Shangguang Wang. *A novel digital twin-centric approach for driver intention prediction and traffic congestion avoidance*. Journal of Reliable Intelligent Environments, 2018.

Yuheng Du and Mashrur Chowdhury. *A Distributed Message Delivery Infrastructure for Connected Vehicle Technology Applications*. IEEE Transactions on Intelligent Transportation Systems, 2017.

Amazon Web Services. What is amazon ec2?, 2022b. URL https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

Maxime Guériau and Ivana Dusparic. Quantifying the impact of connected and autonomous vehicles on traffic efficiency and safety in mixed traffic. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8, 2020. doi: 10.1109/ITSC45102.2020.9294174.

German Aerospace Center. Inductive loop value retrieval, 2022c. URL https://sumo.dlr.de/docs/TraCI/Induction_Loop_Value_Retrieval.html.

Deng S Yu, S. Adaptive vehicle extraction in real-time traffic video monitoring based on the fusion of multi-objective particle swarm optimization algorithm. 145, 2018. doi: https://doi.org/10.1186/s13640-018-0381-8.

Docker. Docker overview, 2022a. URL https://docs.docker.com/get-started/overview/.

Docker. Get started with docker compose, 2022b. URL https://docs.docker.com/compose/gettingstarted/.

Apache Kafka. Release notes - kafka - version 2.5.0, 2020. URL https://archive.apache.org/dist/kafka/2.5.0/RELEASE_NOTES.htmll.

Python.org. Python json documentation, 2022. URL https://docs.python.org/3/library/json.html.

German Aerospace Center. Sumo user documentation - controlling the same simulation from multiple clients, 2022d. URL https://sumo.dlr.de/docs/TraCI/Interfacing_TraCI_from_Python.html#controlling_the_same_simulation_from_multiple_clients.

A1 Appendix

```
{
    'loop_id': 'M50_Northbound',
    'lane 1': '0',
    'lane 2': '0',
    'lane 3': '0',
    'lane 4': '1',
    'timestamp': '2022-04-05 0:00:05'
}
```

Figure A1.1: Typical inductive loop message

```
{
    'lane_id': '4414080#1-AddedOnRampEdge.343',
    'lane_index': '0',
    'direction': 'N',
    'distance': '94.39',
    'speed': '91.67',
    'class': 'passenger',
    'timestamp': '2022-04-05 0:01:56'9
}
```

Figure A1.2: Typical toll bridge message

```
{
    'camera_id': 'M50(S) After J9 (N7)',
    'lane_id': '75259388-AddedOffRampEdge.76',
    'lane_index': '1',
    'direction': 'S',
    'distance': '109.43',
    'speed': '107.02',
    'timestamp': '2022-04-05 0:00:17'
}
```

Figure A1.3: Typical motorway camera message

```
{
    'probe_id': 'NRA_000000001070_Eastbound-2.1',
    'location': '(53.31484349209639, -6.383704806116359)',
    'speed': '78.3',
    'vehicle type': 'CAV4',
    'timestamp': '2022-04-05 0:00:22'
}
```

Figure A1.4: Typical probe vehicle message

	Size (bytes)
Inductive Loops	124
Toll Bridge	190
Motorway Cameras	205
Probe Vehicles	177

Table A1.1: Typical message size in bytes

	A	B	C	D	E	F	G
Inductive Loops	52.13	52.74	52.13	48.08	40.70	50.87	38.25
Toll Bridge	90.47	89.98	90.46	86.88	76.80	88.97	84.09
Motorway Cameras	91.51	95.86	96.51	89.99	80.00	110.00	80.00
Probe Vehicles	150.33	149.95	150.33	137.56	130.00	155.07	159.98

Table A1.2: Latency (ms) by topic (low traffic)

	A	B	C	D	E	F	G
Inductive Loops	41.34	44.98	41.36	41.96	39.30	32.72	40.62
Toll Bridge	149.39	155.03	149.34	145.43	141.12	140.85	159.07
Motorway Cameras	110.00	118.03	110.00	100.02	102.52	120.00	110.04
Probe Vehicles	282.22	299.96	282.22	249.89	262.89	281.83	290.03

Table A1.3: Latency (ms) by topic (high traffic)

	A	B	C	D	E	F	G	Final
Latency (ms)	150.38	136.15	150.38	122.53	121.96	166.60	220.04	205.19

Table A1.4: Latency by configuration (low traffic)

	A	B	C	D	E	F	G	Final
Latency (ms)	272.67	255.77	272.67	208.75	234.03	371.21	302.63	279.41

Table A1.5: Latency by configuration (high traffic)

	Location 1	Location 2
Low Traffic	205.19	144.86
High Traffic	279.41	156.40

Table A1.6: Latency (ms) by distance