Integration of computer engineering and wireless communication for digital power systems

Jingyuan Tu

2020-10-28

Acknowledgement

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam egestas vulputate quam at consequat. Morbi blandit mi eu condimentum congue. Praesent mollis est at urna vestibulum sagittis. Phasellus eleifend, urna iaculis euismod egestas, nisl eros porttitor eros, ac ultrices augue dui sed ipsum. Donec pharetra justo posuere neque convallis accumsan. Nam eget enim ultrices, porta orci ac, volutpat felis. Suspendisse laoreet tellus sed massa convallis interdum.

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam egestas vulputate quam at consequat. Morbi blandit mi eu condimentum congue. Praesent mollis est at urna vestibulum sagittis. Phasellus eleifend, urna iaculis euismod egestas, nisl eros porttitor eros, ac ultrices augue dui sed ipsum. Donec pharetra justo posuere neque convallis accumsan. Nam eget enim ultrices, porta orci ac, volutpat felis. Suspendisse laoreet tellus sed massa convallis interdum.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam egestas vulputate quam at consequat. Morbi blandit mi eu condimentum congue. Praesent mollis est at urna vestibulum sagittis. Phasellus eleifend, urna iaculis euismod egestas, nisl eros porttitor eros, ac ultrices augue dui sed ipsum. Donec pharetra justo posuere neque convallis accumsan. Nam eget enim ultrices, porta orci ac, volutpat felis. Suspendisse laoreet tellus sed massa convallis interdum.

Chapter 2

Literature Review

2.1 Overview

There are many existing solutions and work done by researchers exploring the ways to construct a communication system for monitoring and controlling of solar panels. They can be separated into three categories:

- Wired communication
- Long range wireless communication
- Short range wireless communication

The wired communication requires physical cables to connect the components of the system, the long range wireless communication employs cellular network to transmit data over long distances wirelessly, and short range wireless communication utilises ZigBee to form a mesh network which covers arbitrary area.

While many literatures explored different means of communication for solar power generation system, only a few of them address how the data from solar panels can be processed more timely and efficiently. Related work in other similar fields had been reviewed which suggust the use of message queues and real-time distributed computation systems may improve the performance and usability of such a system.

2.2 Wired communication

The conventional method of monitoring a large scale solar farm is using physical cables. In this setup, a set of power converters read the current and voltage, then transmit the sensing data to the monitoring station over some physical cables. One of the benefits of using physical cables is its proven reliability that had been demonstrated throughout solar power generation systems around the world. Furthermore, physical cables have high bandwidth which enables complicated sensing data with high sampling rate can be transmitted without fully saturating the link [Shariff et al., 2015]. However, the researchers have found the lifespan of the system may be reduced due to physical cables are exposed to constant sunlight and rain [Shariff et al., 2015]. Furthermore, the signals in the cables may attenuates over long distances and counter measures such as repeater may be required.

2.3 Long range wireless communication

One of the long range wireless communication that had been studied is a GSM based communication system for solar powered street light. In this system, each street light has a microcontroller monitoring the power generation, battery status, battery behaviours, and light behaviours. Each microcontroller communicates with a remote server through Short Message Service (SMS), which is the underlying technology that enables mobile phone to send and receive text messages from or to other mobile phones. The SMS messages carries the sensing data from each street light to the remote server through a SMS gateway to be processed and stored in the database. Finally, a web console can be used to visualise the data [Siregar and Soegiarto, 2014]. The use of SMS solves a big issue that other means of communication such as Bluetooth, ZigBee, and Wi-Fi has, the lack of range. With this setup, a large area of street lights can be connected to the SMS gateway thanks to the large communication range of the underlaying GSM communication system. However, the major drawback of using SMS as the underlaying means of communication is the size of each SMS message must be within 160 characters [ets, 1995]. The limitation means the sensing data cannot be very complicated and detailed, which limits the capability of the system and its use cases. The remote server also plays an important role in the communication system. It is composed of a GSM gateway that is connected to a web server, and the web server is exposed to the internet so everyone can access the visualisation of the data over the internet. The web server is also connected to a database such that the sensing data is persistent even if the web server is down [Siregar and Soegiarto, 2014]. Unfortunately, the cost estimate and the performance of such a system, and the underlaying architecture of the web server are not documented in the literature. It is hard to quantitatively compare this solution to other solutions in terms of price and performance. This system is also intended to be used with street lights that have much lower density than solar panels, and the behaviour of such a system is undocumented as the number of microcontroller increases. Therefore, it is uncertain that the GSM based communication system would adapt well in a more demanding situation such as solar farm.

Another long range wireless communication system is a GPRS based communication system for solar panel monitoring and controlling. The system is designed around the concept of Internet of Things (IoT) in mind and it is separated into three layers, the sensing layer, network layer, and application layer. The sensing layer contains sensors monitoring the characteristics of the solar panel. Then, the sensing data are fetched into a microcontroller equipped with a GPRS modem. Finally, the sensing data are sent to the internet through the GPRS modem. The application layer contains advanced functionalities such as data analytics, fault monitoring, generation monitoring, and functionalities that leverage the powerful processing capabilities of the server. Between the sensing layer and application layer, the network layer bridge the two layers by providing internet access and hosting database for persistent data storage [Adhya et al., 2016]. The system utilises the GPRS cellular network, which is the primary means of communication between mobile phones in the 2000s. This technology have the benefit of covering a wide area and offers internet access with limited speed. That means, a single GPRS cellular tower can cover significantly more solar panels comparing to other wireless technologies such as Bluetooth, ZigBee and Wi-Fi. Having direct internet access also means it doesn't have to translate the sensing data from one communication system to another like the GSM based system, where data are being sent through SMS messages and translated at the GSM gateway into HTTP messages. Unfortunately, GPRS is still very slow by today's standards with only 14kbps upload [3gp, 2020]. It imposes a substantial limit on the complexity and amount of data being sent to the server for processing. More importantly, countries worldwide had been planning on decommissioning this old means of communication before mid 2020s. Therefore a new carrier for the sensing data needs to be used. Similar to the previous literature, there is no cost estimation of such a system and no performance details were documented.

2.4 Short range wireless communication

A short range communication method that are evolving around the IoT concept is ZigBee which is explored over the recent years. Like many other communication systems, there needs to be a ZigBee gateway with internet access to forward data from the solar panels to the server and persistent data store using HTTP protocols. Furthermore, the limited communication range of ZigBee allows the the power consumption to be is very low. The researchers claim a ZigBee module can be powered by non-rechargeable battery for two to three years and it can connect up to 65000 ZigBee modules [Shariff et al., 2015]. Since ZigBee is designed for modern IoT applications, it has more than sufficient bandwidth of up to 250kbps for complicated data exchange over the network. The problem with ZigBee is its limited range of around 1500 meters with direct line of sight and the range may be reduced significantly if there are obstructions [Shariff et al., 2015]. To combat this issue, ZigBee modules have the capability to form a cluster of connected ZigBee network which allows it to cover a much wider area. The supported network topologies of ZigBee modules are Star, Cluster Tree, and Mesh Network [Shariff et al., 2015].

The star topology has a primary ZigBee module that are directly connected to all secondary ZigBee modules. Since they are directly connected to the primary module, the resources of the secondary modules such as bandwidth are completely reserved for its own communication with minimal latency. This topology doesn't increase the coverage of a ZigBee Network but it is suitable for a small cluster of solar panels to communicate with the gateway that has internet access such as roof top solar systems [Shariff et al., 2015].

The cluster tree topology is formed by connecting many networks with star topology. The sensing data from each module are sent and forwarded to the gateway through arbitrary number of ZigBee modules assuming there exists at least one gateway within the network that are directly or indirectly reachable from the originating module. This topology enables arbitrary coverage of the communication system. The problem with this topology is some ZigBee modules that needs to forward many other modules' sensing data may easily overwhelm its available bandwidth, leading to performance degradation [Shariff et al., 2015].

Finally, the mesh topology allows each ZigBee module having more than two directly connected modules, which allowing them to form a mesh network. Similar to the cluster tree topology where the coverage is arbitrary, it also has the benefit of reducing the potential to reach bottleneck as the sensing data can be sent through many different path [Shariff et al., 2015].

2.5 Computation system

Since the lack of studies with regards to the computer system architecture used in the communication system for solar panels. System architectures for communication systems that requires similar capabilities are reviewed instead. One of the proposed system is designed for monitoring automotive manufacturing processes using IoT devices, and it incorporates Kafka, Apache Storm, and MongoDB to process, monitor, and store data in real-time [Syafrudin et al., 2018]. In this architecture, Kafka is being used as a message queue for asynchronous communication between components of the system and it employs the publisher subscriber design pattern. The pattern separates the responsibility of the system into producer and consumer where a producer generates data, assign it with a topic that consumer can subscribe to, and consumers are notified when a new piece of data from their subscribed topics is available. The Apache Storm is used for real-time processing. It is subscribed to the topics in the Kafka and react to those new data depending on the type of sensor. In general, it stores the new data into MongoDB, a persistent data storage, and publish new data to the user so data is available as soon as it is generated and processed. The advantage of this architecture is the user can view the data in real-time whenever it is ready, whereas many other simple architectures consists of a single HTTP web server requires the user to refresh the website constantly to download new data. This enables the user to make timely decisions, and reduces the workload of the server as it only processes the data when it is available and the user doesn't querying for data constantly.

2.6 The missing piece of the puzzle

Currently, most literatures have been strongly focusing on the means of communication between the solar panels and gateway, or the computation systems that are processing the data in the background. There is barely anywork done that investigates the union of the both worlds, that is, investigating a real-time computation system that maximises the benefits of the selected means of communication with respect to the solar power generation systems. In my opinion, this is one of the most important piece of the puzzle that is missing within this problem space because:

- Lack of documentation regarding the performance meaning we cannot quantitatively compare, justify, and identify the areas of improvement.
- Lack of integration between the two research areas meaning we cannot predict if the selected combo would work together efficiently no matter how promising they are in its own right.
- Lack of cost estimate of the system meaning we have no way to tell if the proposed solution is practical and feasible.

Chapter 3

System Overview

3.1 Concept

3.1.1 Monitoring

The solar panel generates electricity and feed it into the converter after being measured by the current and voltage sensor. The converter converts the electricity from a higher voltage to a desired votlage, which can be measured again by the sensor and used to charge the battery. The sensing data consist of input current and voltage, and output current and voltage, which are fed into the microcontroller. The microcontroller has a 4G cellular modem which enables communication with the surrounding 4G cellular base station, and the internet access is provided by the gateway at the base station. That means, the sensing data can be be sent from the microcontroller to the server through the 4G cellular network. Once the sensing data reaches the server, the data is processed, store into the database, and send to the client such as a web application. The concept for monitoring is shown in the figure 3.1.

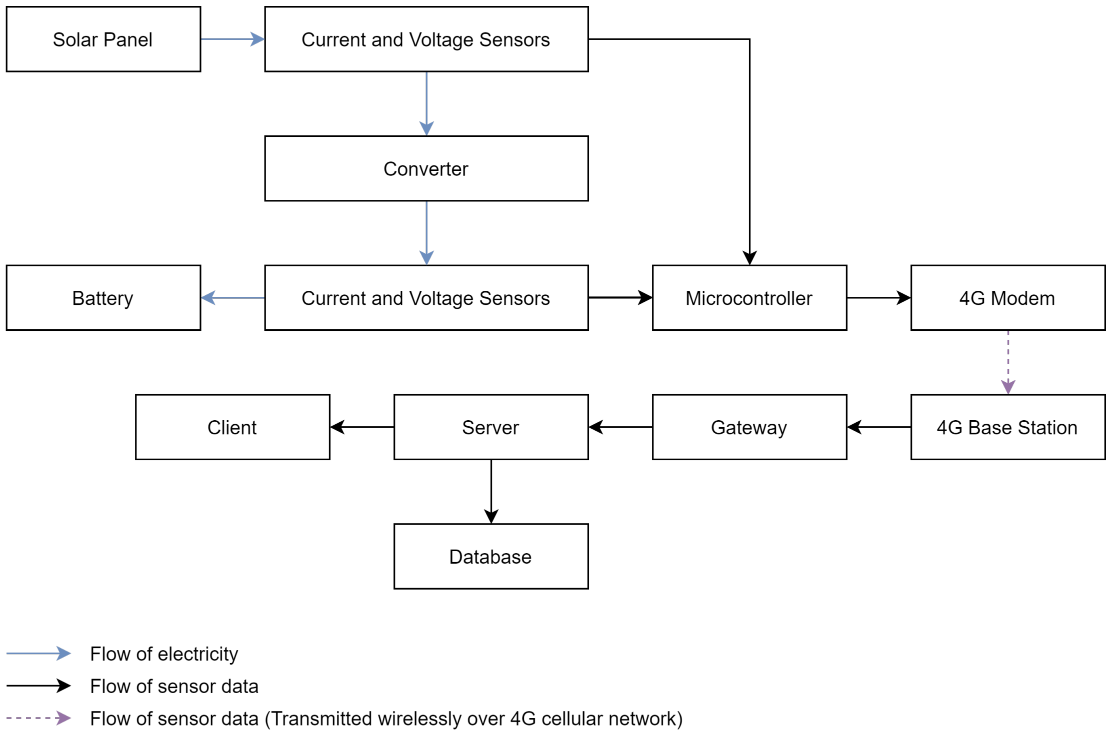

Figure 3.1: Concept of the proposed system for monitoring.

3.1.2 Controlling

The lifecycle of a command begins at the client initiated by the user. The command can be anything from setting the output voltage, to shutting down the converter. Once initiated, the client contact the server to let it know a command had been issued, and the server store it in the database.

On the other hand, the microcontroller performs periodic polling, that is, querying the server if there are new commands issued by the user every second. Similar to the monitoring, the query that is sent from the microcontroller to the server and the reply of the server that is consisting of a list of new commands are transmitted over the 4G cellular network. Finally, the commands are received by the microcontroller and the changes are applied to the converter. The concept for controlling is shown in the figure 3.2.

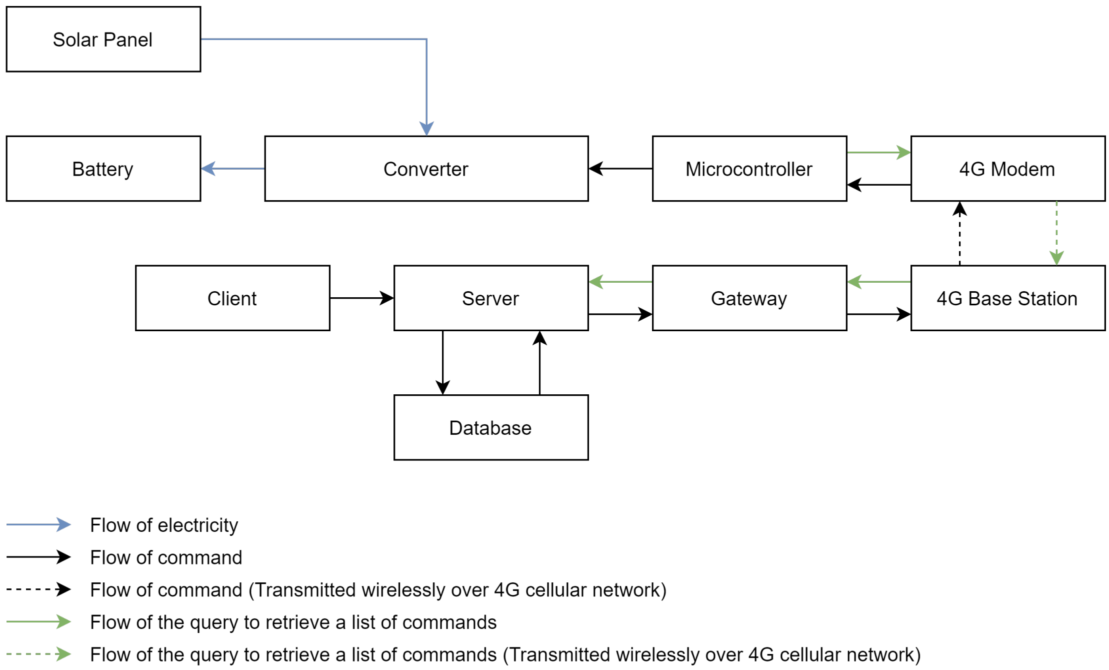

Figure 3.2: Concept of the proposed system for controlling.

3.2 Layers and components

The proposed system is composed of five layers separated by responsibilities. They are:

- Storage Layer
- Server Layer
- Relay Layer
- Simulation Layer
- Sensing Layer

The storage layer is responsible for storing the state of the server and sensing data received from the microcontrollers. The server layer is where the data processing logic, system monitoring, and data visualisation services are located. The relay layer contains the 4G infrastructures and providing internet access to 4G enabled devices. The sensing layer is responsible for measuring the data and control the generation of power. Lastly, the simulation layer is used to simulate the behaviours of the sensing layer for benchmarking purposes. Figure 3.3 shows the layers and the components within each layer.

3.3 Responsibilities

In this section, the responsibilities of each component are described.

3.3.1 Storage layer

On-disk database

The distinguishing characteristic of this type of database is the data that are stored in the database is persistant, which means the data remain intact even if the database is down due to failures. It is usually used to store a large amount of data reliably. The downside of using an on-disk database is the response time of querying and inserting data into the database can be very slow. The primary usage in the proposed system is storing the information, status, and sensing data of a device such as a solar panel persistently.

In-memory database

The goal for this type of database is not storing the data persistently but aiming to access, modify, and insert data as quick as possible. This advantage is the result of using the memory instead of the disk of the server, which is significantly faster. However, it also comes with the problem of having significantly smaller capacity, and the data is volatile. That is, the data is gone when the database

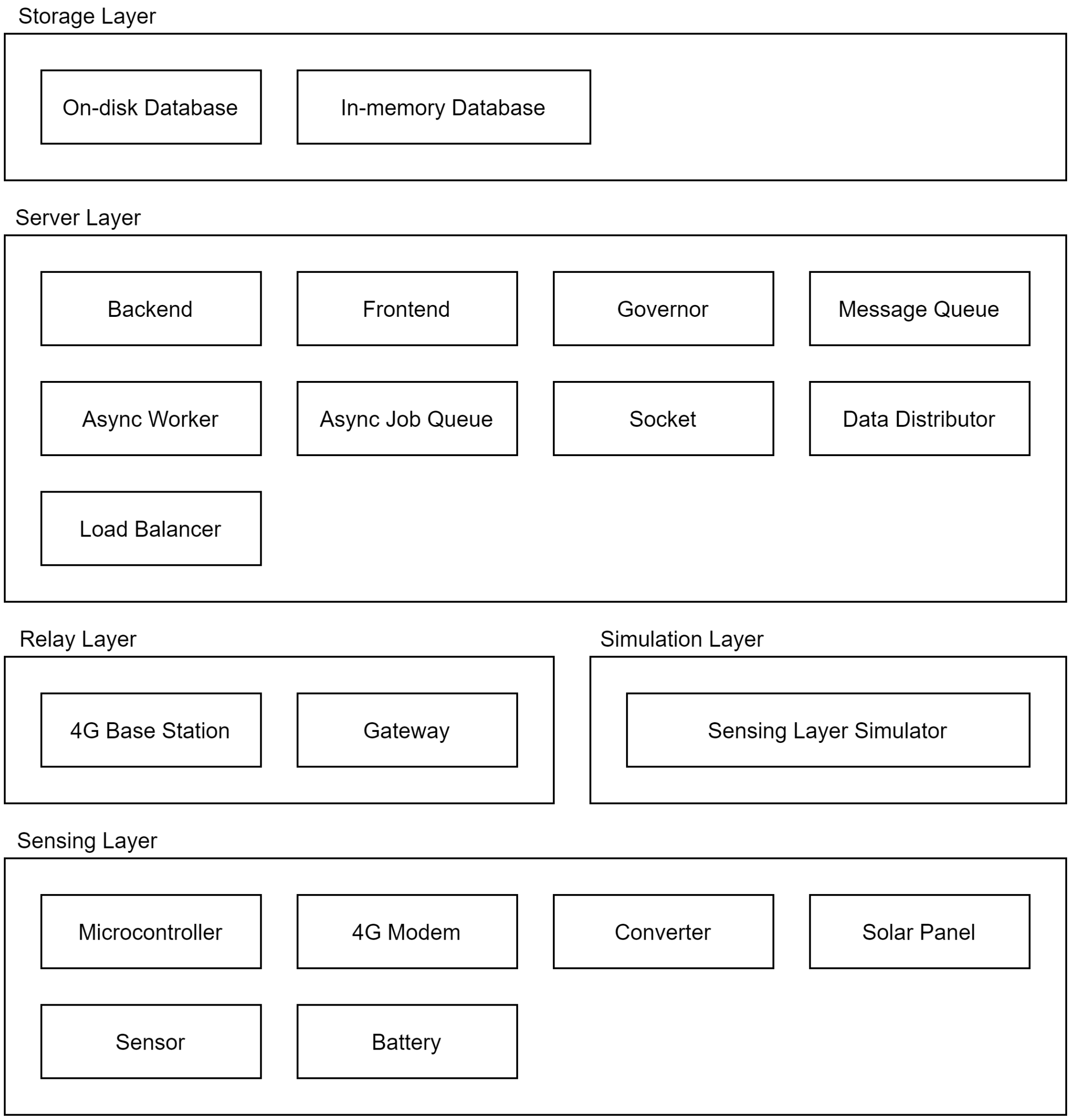

Figure 3.3: Layers and components of the proposed system.

is down and the data may be purged if the memory of the server is running low. This is primarily used to synchronise the state and sharing data between backend instances that are running on the same server.

3.3.2 Server layer

Backend

Backend is a stateless HTTP server and its responsibilities are:

- Providing application programming interface (API) for microcontrollers to register itself, insert sensing data to the database, and retrieve issued commands.
- Providing API for frontend to retrieve data for visualisation and notifying a command had been issued.
- Providing context-aware load balancing for frontend's real-time data visualisation.

Frontend

Frontend is a web application and its responsibilities are:

- Providing a pleasant and usable user interface.
- Providing a real-time data visualisation.
- Providing a control panel for issueing commands.

Load balancer

It is a piece of software that distribute the incoming requests to the server instance with the least amount of load. It also uses multiple threads to serve the frontend web pages to improve the performance under load.

Governor

The governors are daemons for house keeping. That is, they are processes that are running in the background of the server and performing some predefined tasks periodically. There is only one governor in the proposed system, its called the data aggregation governor and its job is aggregating the real-time data every minute.

Message queue

This is an asynchronous communication system between the components of the server layer. The main contribution of the message queue is enabling processes such as backend, governor, and data distributor to communciate with each other despite they are completly separate programs.

The message queue is asynchronous because the originating process can send a message and continue to perform other tasks without waiting for the message to arrive or a reply is received. As oppose to synchronous communication, where the originating process would stop doing any work and waiting for a reply to receive before continuing.

Async job queue and worker

The asynchronous job queue is a queue of tasks to be executed in a first in first out fashion, and the worker is a thread that takes a job from the queue in order and actually execute them. The backend is using this mechanism to process lengthy but not time sensitive tasks such as inserting data into the database to improve performance. Similar to the message queue, the originating process adds a job to the queue and it continues to perform other work without waiting for the work to finish.

Socket

Socket is the key to efficient real-time communication between the backend and the frontend. This is the enabling tehcnology which allows the frontend to wait for new data to be available as opposed to letting the clients asking for new data every second, which is very inefficient.

Data distributor

The data distributor sits between the clients and the backend during the real-time communication. Its purpose is reducing the load of the server by publishing the data to the relevant clients directly.

Without the data distributor, the clients would be notified when a new data is available, and all clients would query the backend for the same piece of data, causing the backend to be accessed directly proportional to the number of clients. By using the data distributor, the data is distributed directly to the clients and no backend access is needed. Note that the data distributor also knows what data each client want and only distribute the data to the clients that actually interested in the data.

3.3.3 Relay layer

This layer contains two key infrastructures, the 4G base station, and the internet gateway. The 4G base station allows 4G enabled devices to communicate with it and forward the data to its internet gateway so that devices can access the internet wirelessly.

3.3.4 Sensing layer

This layer contains power generation devices such as solar panels, power converters, and batteries in our experimental setup. There are also monitoring devices such as sensors that are attached to the input and output of the converter and sends analog signals to the microntroller. Then, the microcontroller process those signals and send the data to the server over the 4G cellular network using the 4G modem.

3.3.5 Simulation layer

This layer is used for benchmarking as it can generate any number of virtual solar panels and push simulated data to the server. This enables the performance and characteristics of the system to be measured and this layer should behave identical to the sensing layer from the server layer's point of view.

3.4 Hardware

The table 3.1 outlines the hardware that had been used in the system. Details of each hardware can be found in the subsections below.

Table 3.1: List of hardware in the proposed system.

Type	**Hardware Model**
Benchmarking server	Custom Built Desktop (See subsection 3.4.1)
Deployment server	AWS EC2 T2-Medium
Microcontroller	Nuvoton M263A
4G Modem	Quectel EC25
4G Antenna	Antenova SRFL029
4G Base Station	Depending on the service provider
Gateway	Depending on the service provider
Converter	Morningstar PS-MPPT-40M
Sensor	ACS723 and ACS724
Battery	Sun Xtender PVX-890T

3.4.1 Benchmarking server

This is the server that is being used to benchmark the proposed communication system. The technical specification is detailed in the table 3.2.

Table 3.2: Technical specification of the benchmarking server.

Name	**Specification**
CPU	AMD Ryzen 3800X (8 Core 16 Threads at 4.3 Ghz)
GPU	Nvidia GTX 1080Ti
Memory	16GB 3200Mhz
Disk	4TB HDD + 512GB SSD

3.4.2 Deployment server

This is the server that is hosted on the AWS cloud computing platform and it is used to deploy the proposed system so that it can be accessed publically. The technical specification is detailed in the table 3.3. Note that it is not neccessary to deploy the proposed system to a powerful cloud server as there is only one solar panel used for real-world testing.

Table 3.3: Technical specification of the deployment server.

Name	**Specification**
Instance Type	T2.Medium
CPU	2 Cores
Memory	4GB
Disk	32GB

3.4.3 Microcontroller

The selected microcontroller is Nuvoton's M263A. The critical features for this microcontroller that are relevant to the proposed system is shown in the table 3.4. See figure 3.4 and 3.5 for additional details of the microcontroller.

3.4.4 4G modem

The selected LTE module is Quectel EC25. The technical specification is shown in the table 3.5 and the figure 3.6 shows the LTE module.

3.4.5 Converter

The converter used in the real-world testing is the Morningstar PS-MPPT-40M. The technical specification is outlined in the table 3.6. Note that the proposed

Table 3.4: Critical features of Nuvoton M263A

Criteria	**Specification**
CPU	ARM Cortex M23 64MHz
APROM Memory	512KB
SRAM Memory	96KB
LDROM Memory	4KB
Has PWM Pins	True
Has Analog Pins	True
Has SIM card slot	True
Has 4G modem support	True

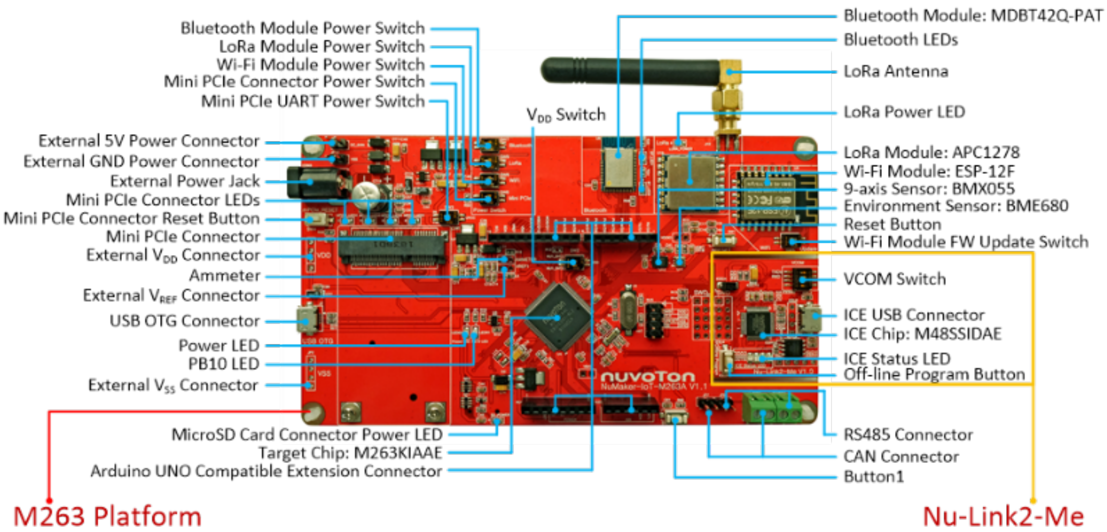

Figure 3.4: Nuvoton M263A (front) [Nuv,]

Table 3.5: Technical specification of Quectel EC25

Feature	**Specification**
Form factor	mini PCIe
Downlink	150Mbps
Uplink	50Mbps
Has LTE	Enabled

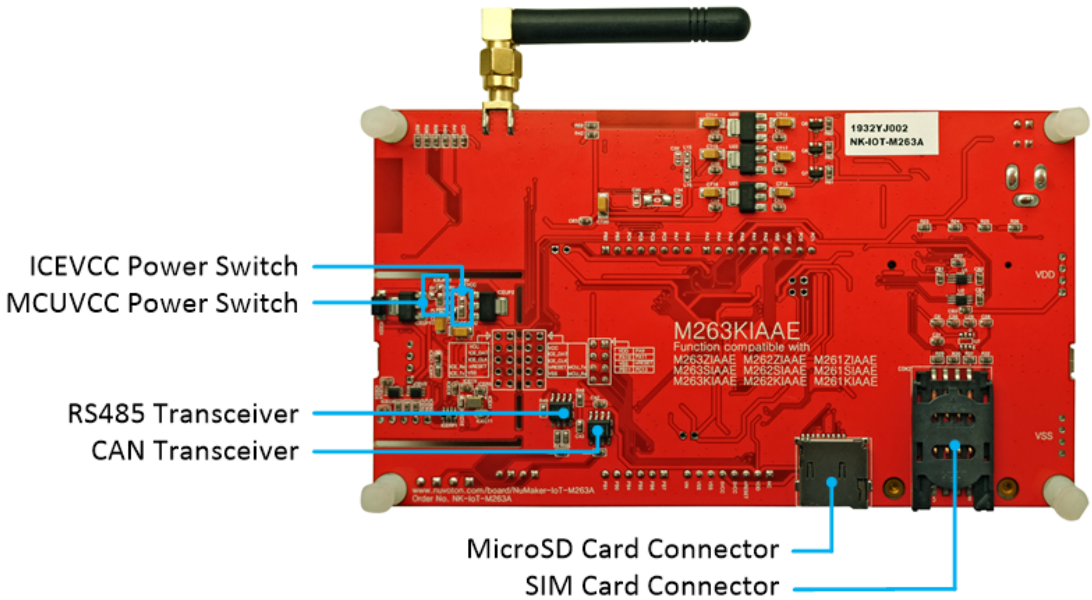

Figure 3.5: Nuvoton M263A (back) [Nuv,]

Figure 3.6: Quectel EC25 [Que,]

system is designed to be generic such that any converter would work similarly with the system.

Table 3.6: Technical specification of Morningstar PS-MPPT-40M

Feature	**Specification**
Max Battery Current	40 Amps
Load Current Rating	30 Amps
Nominal Battery Voltage	12V or 24V
Peak Efficiency	98%
Battery Voltage Range	10-35 V

3.4.6 Voltage and current sensor

There are two sensors that are being used to read the voltage and current. The ACS723 shown in the table 3.7 is used to measure the input voltage and current, while ACS724 shown in the table 3.8 is used to measure the output voltage and current. Note the input voltage and current are referring to the voltage and current of the electricity flowing from the solar panel to the converter, and the output current and voltage are referring to the voltage and current of the electricity flowing from the converter to the battery. Additionally, the proposed system should work with any sensor with analog output as long as the parameters in the microcontroller are properly set.

Table 3.7: Technical specification of ACS723 sensor

Characteristic	**Specification**
Sensitivity	Typ. 400 (mV/A)
Zero-Current Output Voltage	Typ. VCC * 0.1 (V)

Table 3.8: Technical specification of ACS724 sensor

Characteristic	**Specification**
Sensitivity	Typ. 100 (mV/A)
Zero-Current Output Voltage	Typ. VCC * 0.5 (V)

3.4.7 Battery

The battery model used in the real-world testing is Sun Xtender PVX-890T. The technical specification for the battery is shown in the table 3.9.

Table 3.9: Technical specification of Sun Xtender PVX-890T

Characteristic	**Specification**
Nominal Voltage	12 Volt
Ampere Hour Capacity @ 24 Hour Rate	89
Bulk/Absorb Charge	14.2-14.4 Volts
Float Charge	13.2-13.4 Volts

3.5 Software

The table 3.10 outlines the software that had been used in the system and their responsibilities are outlined in the section 3.3.

3.6 Cost estimate

The cost of the microcontroller that is being used in the thesis is shown in the table 3.11. Note that there are many other modules such as Wi-Fi, LoRaWAN, Bluetooth, accelarometer, temperature sensor, etc. included in the price of the microcontroller which are not necessary for the proposed system to work. Addi-

Table 3.10: List of software in the proposed system.

Component	**Software**
Benchmarking operating system	Ubuntu Desktop 20.04 LTS
Deployment operating system	Ubuntu Server 20.04 LTS
Microcontroller platform	Mbed OS 15.5
Message queue	Apache Pulsar
Job queue and worker	RedisRQ
Frontend	ReactJS and nivo.rock
Backend	Python Flask
Socket	socket.io
Load Balancer	nginx
WSGI Server	Gunicorn
On-disk Database	MongoDB
In-memory Database	Redis
Programming language	Python 3.6

tionally, the cost listed in the table is based on the retail price not the wholesale price.

Table 3.11: The cost of the microcontroller in the experimental setup.

Component	**Retail price**
EC25 LTE Modem (mini PCIe)	69 AUD
Nuvoton M263A	143 AUD
Antenova SRFL029	10 AUD
ACS723 Sensor	16 AUD
ACS724 Sensor	14 AUD
Total	252 AUD or 177 USD

The table 3.12 shows a rough cost estimate for a microcontroller that supports LTE cellular netowork, PWM output, analog input, and preferrably 50Mhz or higher processor clock speed. The cost estimate only shows the cost of the critical components of the microcontroller and miscellaneous cost is not included as they are design dependent.

Table 3.12: Cost estimate of a microcontroller that meets the minimum requriements.

Component	**Retail price**
Sierra HL Series LTE Modem (LLC)	22 AUD
M2351KIAAE Processor	5 AUD
Antenova SRFL029	10 AUD
ACS723 Sensor (LLC)	6.5 AUD
ACS724 Sensor (LLC)	4.5 AUD
Total	48 AUD or 33 USD

Chapter 4

System Architecture and Behaviours

4.1 Introduction

In the section 3.2 and section 3.3, the logical layering and the responsibilities of the components were introduced but how they fit together and work in harmony is still a mystery. In this chapter, I will put the pieces of the puzzle together by discussing of the actual architecture of the system and its behaviours with increasing levels of detail.

4.2 Top-level architecture

The top-level architecture is a simplified architecture where many low-level components are being grouped together and treated as a singular composite component. It is shown in the figure 4.1. The following components in the top-level architecture are composite component:

- Remote Sensing (See subsection 4.4.1)

- Backend (See subsection 4.4.2)
- Frontend (See subsection 4.4.3)
- Real-time (See subsection 4.4.4)
- Governor (See subsection 4.3.3)

4.3 Top-level behaviours

This section provides a general description of how the the proposed system works together with many low-level details abstracted.

4.3.1 Initialisation

The initialisation process is referring to a device in the remote sensing composite component such as a microcontroller attempting to connect with the server before it can send data and receive commands. When a microcontroller is initialising, it sends a register device request to the backend and then wait for the reply from the backend. Once the request is received by the backend, it tries to find a matching device in the database. If found, then it updates the device status to online. Otherwise, it generates a new device ID, creates a representation of the new device in the database, and set the device status to online. Finally, the backend notifies the microcontroller by returning a response to the initial device registration request, and the intialisation sequence is complete. The figure 4.2 shows the flow of the initialisation process and the relevant function calls shown in the top-level architecture.

The initialisation process achieves two goals, the first goal is letting the backend knows the device is online and ready to transmit data, and the second goal is

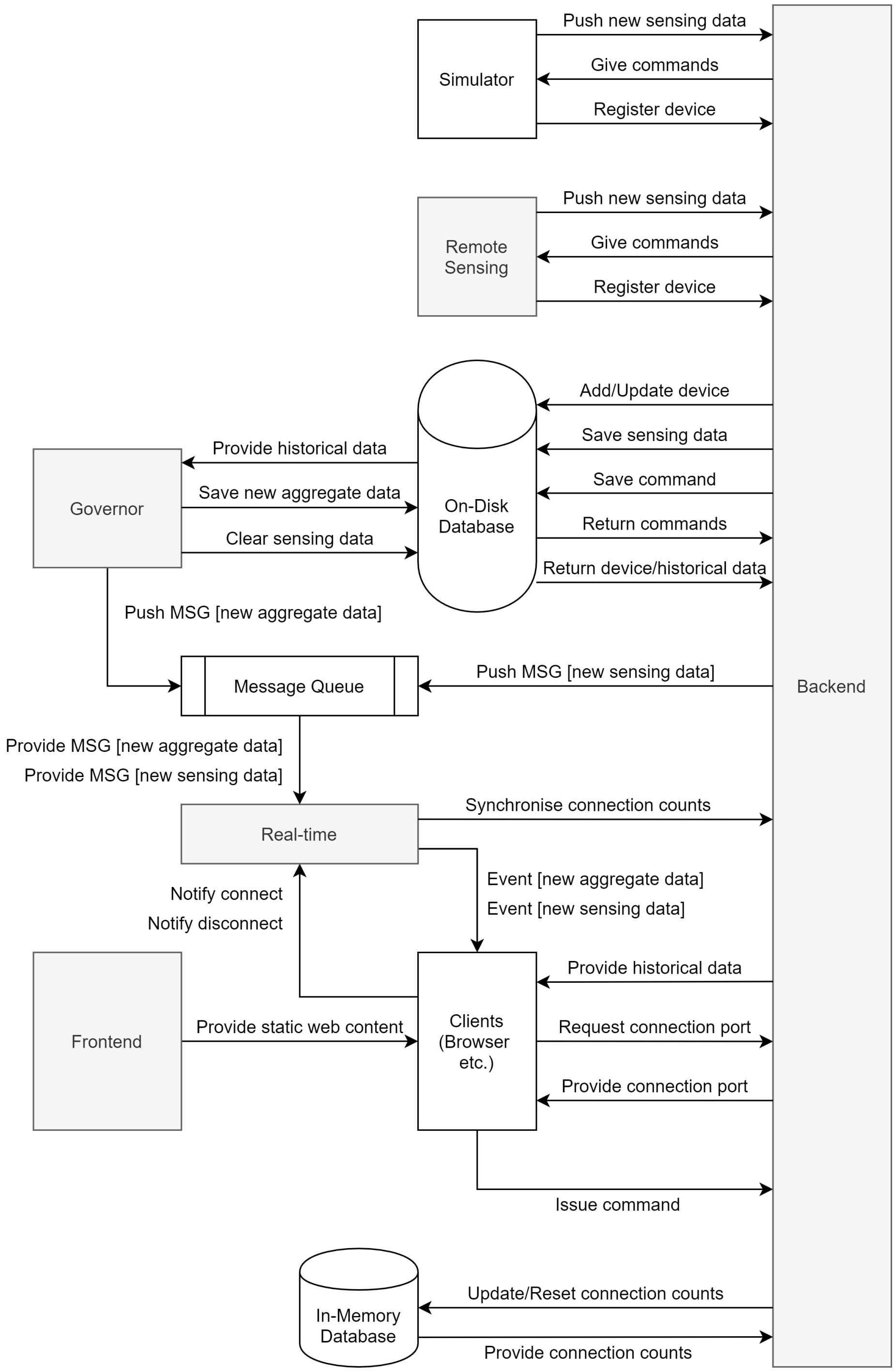

Figure 4.1: Top-level architecture of the proposed system.

getting a unique ID so that it can be identified during the data transmission phase. Once the initialisation process is finished, the microcontroller is ready to transmit data.

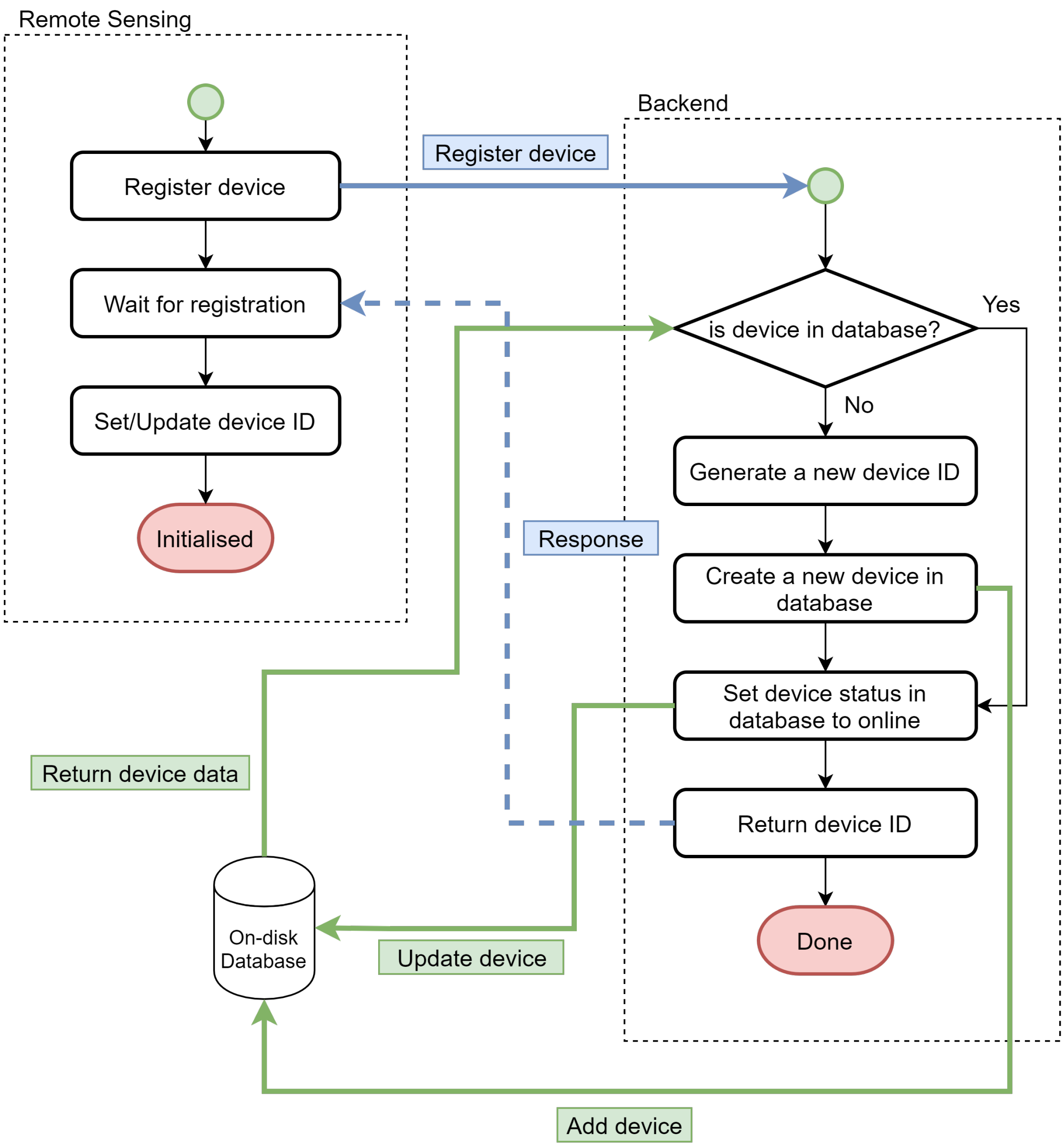

Figure 4.2: The initialisation process.

4.3.2 Data recording and device controlling

The data recording process aims to record data from the sensors and store them into the database. The process begins at the microcontroller where it reads the sensing data from the sensors and then sends it to the backend using the *push new sensing data* request. Once the backend received the sensing data, it begins executing two sequences of instructions simultaneously. The first sequence is used for data recording, where the data is added to the database and then sends a message to clients notifying a new sensing data is available. The second sequence is used for device controlling, where it retrieves a list of commands cumulated in the database, and sends them to the microcontroller through the response to the *push new sensing data* request. As soon as the microcontroller received the commands, it would perform each command in order and wait for 1 second before reading and sending data again. The process is shown in the figure 4.3.

The reason for data recording and device controlling happening at the same time is improving communication efficiency and reducing server load. The microcontroller doesn't need to send two different requests to push data and retrieve commands, which reduces the communication time by half and reduces the number of access to the backend by half.

Executing two seqeunces of instructions simultaneously has the benefit of reducing the response time of a request. The detailed explanation is provided in subsection 4.4.2.

4.3.3 Governing

Governors are a series of background tasks that are executing periodically and making changes to the system. Currently, there is only one governor in the system, the data aggregation governor.

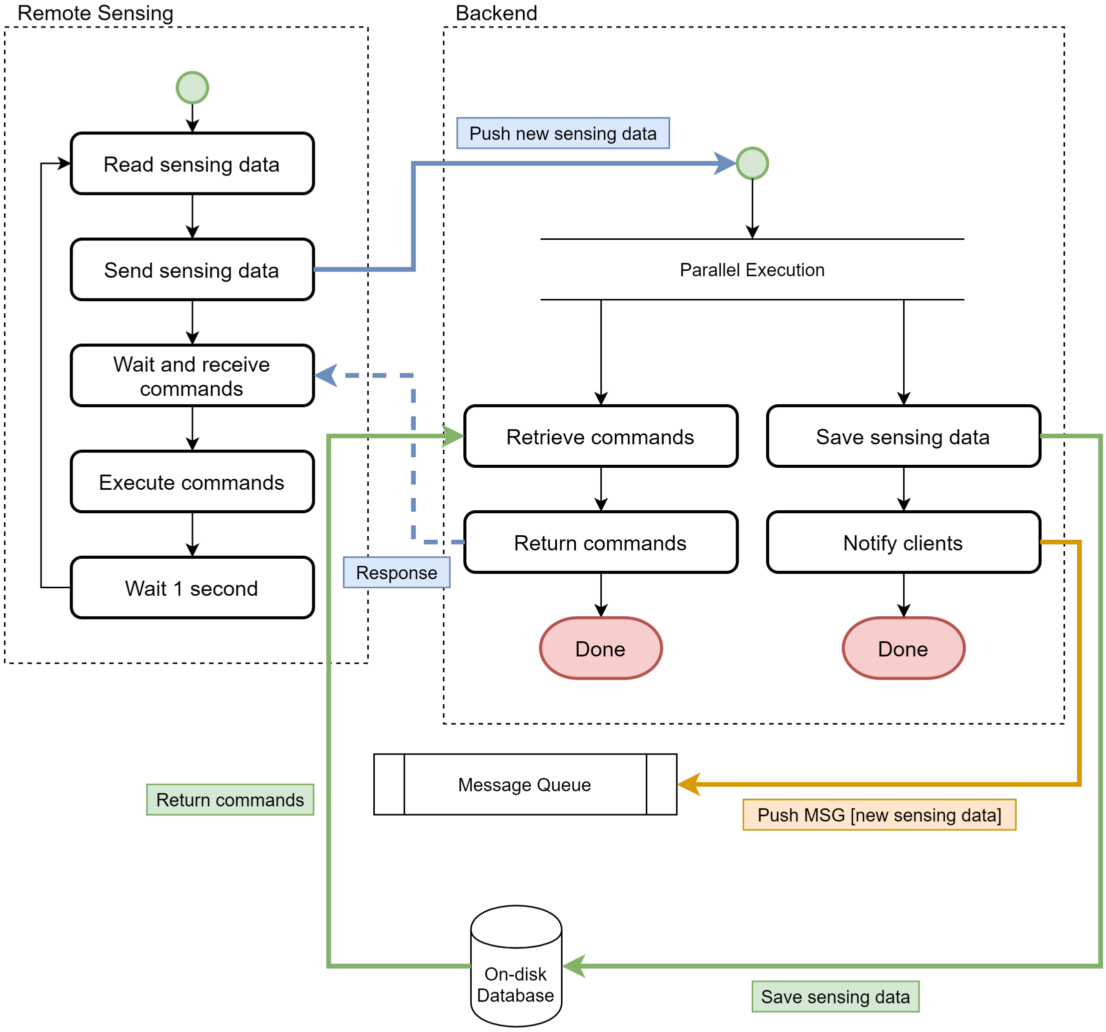

Figure 4.3: The data recording and device controlling process.

Every minute, the governor aggregates the sensing data[1] for every single device by averaging the sensing data received over the last minute, saving it to the database, and notifying clients about a new aggregate data is available. To prevent the database using too much disk space, it resets the sensing data after aggregation so the number of samples in the sensing data is kept below 60 samples. The governing process is shown in the figure 4.4.

[1]The sensing data is also known as the real-time data since it is recorded every second.

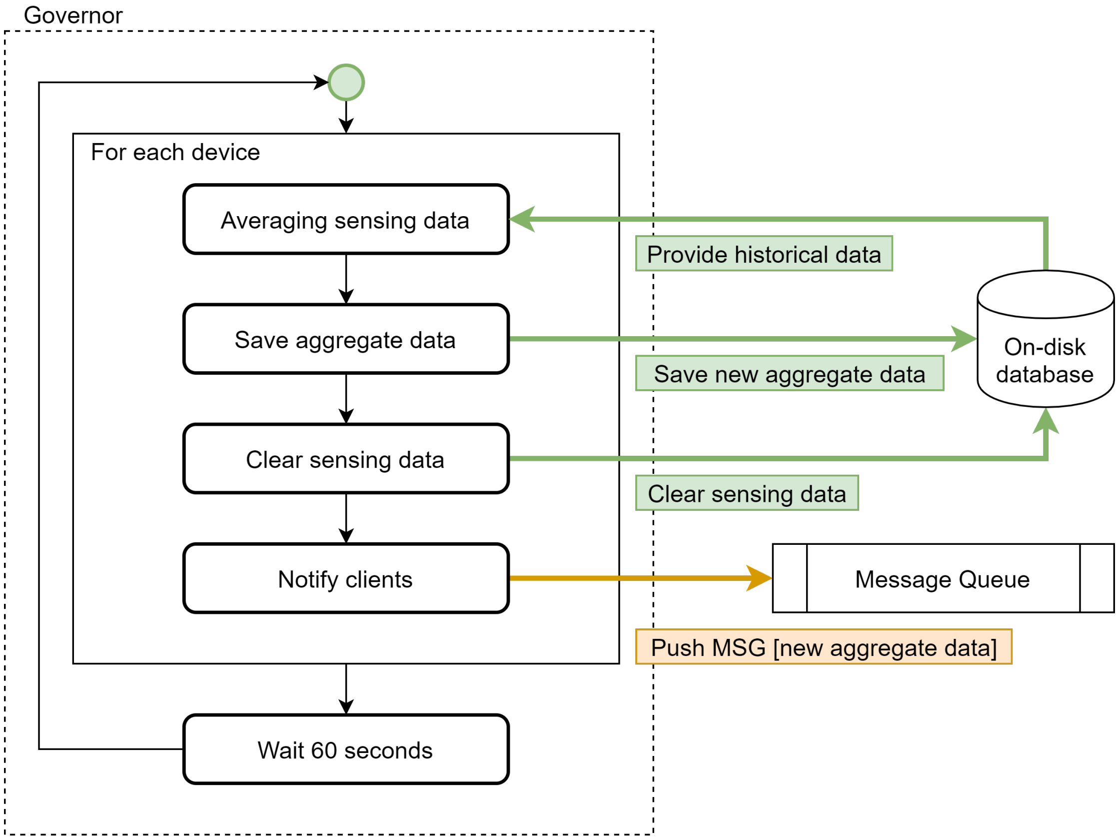

Figure 4.4: The governing process.

4.3.4 Context-aware load balancing

Load balancing is an act of distributing load to different servers evenly to improve the overall performance. This is because it prevents the situation where a single server is overwhelmed by the load while other servers are experiencing starvation[2]. It is context-aware in the proposed system because the load balancer keeping track of the number of active connections a server has and direct the load to the server with the least amount of active connections.

This is different to the conventional load balancers where it uses simple scheduling algorithms such as round-robin to dictate which server the load should be dis-

[2]Starvation in the context of computer system is a resource such as computation power that is being significantly underutilised.

tributed to. The primary reason for using a custom scheduling logic is because the connections in the proposed system are persistent[3] while the connections in a typical system are one-off[4]. That means, if a server is unfortunate enough that all of its active connections have very long lifespan, while connections in other servers have very short lifespan. The simple scheduling algorithm would continue to distribute the load to the server that still has many active connections due to the algorithm only cares about the order of the distribution not the actual load the server has. By being context-aware, the load balancer only distribute the load to the server with the least amount of active connections and the scenario above is avoided.

Back to the proposed system, the context-aware load balancing process is shown in the figure 4.5. For a connecting client, it starts by querying the backend for a port it should connect to[5]. Then, the backend retrieves the connection count for each port from the in-memory database, find the one with the least amount of active connections, and return the port to the client. Once the client knows the port, it connects to the server at the port (the real-time composite component) and notify the server that it has connected. The server then realise it is a connect event and increment its internal connection count which keeps track of its own number of active connections. Finally, the server tells the backend that its active connection count has increased so that the backend is aware of the change and update the connection count associated with the port in the in-memory database.

For a disconnecting client, which is caused by closing the web application or disconnected due to netowork issues. The disconnection would be detected by the server automatically and decrease the internal connection count. Like before, the server tells the backend that a connection has disconnected and the count

[3]The connection may last for a long time, and making many queries while it is connected.

[4]The connection only query the system once and only once.

[5]A server usually sits behind a port in the computer system. Therefore, querying for a connection port is the same as querying for a particular server it should connect to.

is decreased. The backend reacts to it by also decreasing its connection count associated with the port that are stored in the in-memory database.

Notice that both connecting and disconnecting would trigger a connection count update in the backend, and this allows the backend to keep track of the individual server load and distribute the load properly.

The reason for using an in-memory database as oppose to storing the connection count in the server instance is because there are more than one backend instances in the proposed system. If all of them store the connection count in their own instance, then the connection count would not be updated correctly as the connection count synchronisation request is randomly distributed to any of the instances. As a result, each backend instance has a different connection count. Therefore, they need to share the connection count data using the in-memory database so all instances are accessing and updating the same connection count.

4.3.5 Data monitoring

The data monitoring process is about showing the data on the client and the process is shown in the figure 4.6. Once the client is connected to a real-time server, the data monitoring process can begins. The first step is preloading the historical data from the backend which allows the user to view the previously recorded data for upto 4 hours immediatly after the client establishing the connection. During data preloading, the backend is fetching the data from the on-disk database and provide it to the client in the required format. Once the data preloading is done, the client starts listening for incoming data from the real-time server. Recall the new aggregate data is generated by the governor and the new sensing data is generated by the microcontroller, both of them produce a message into the message queue signifying a new piece of data is available. The consumer of both messages is the real-time server where it listens for the new data messages from the message queue

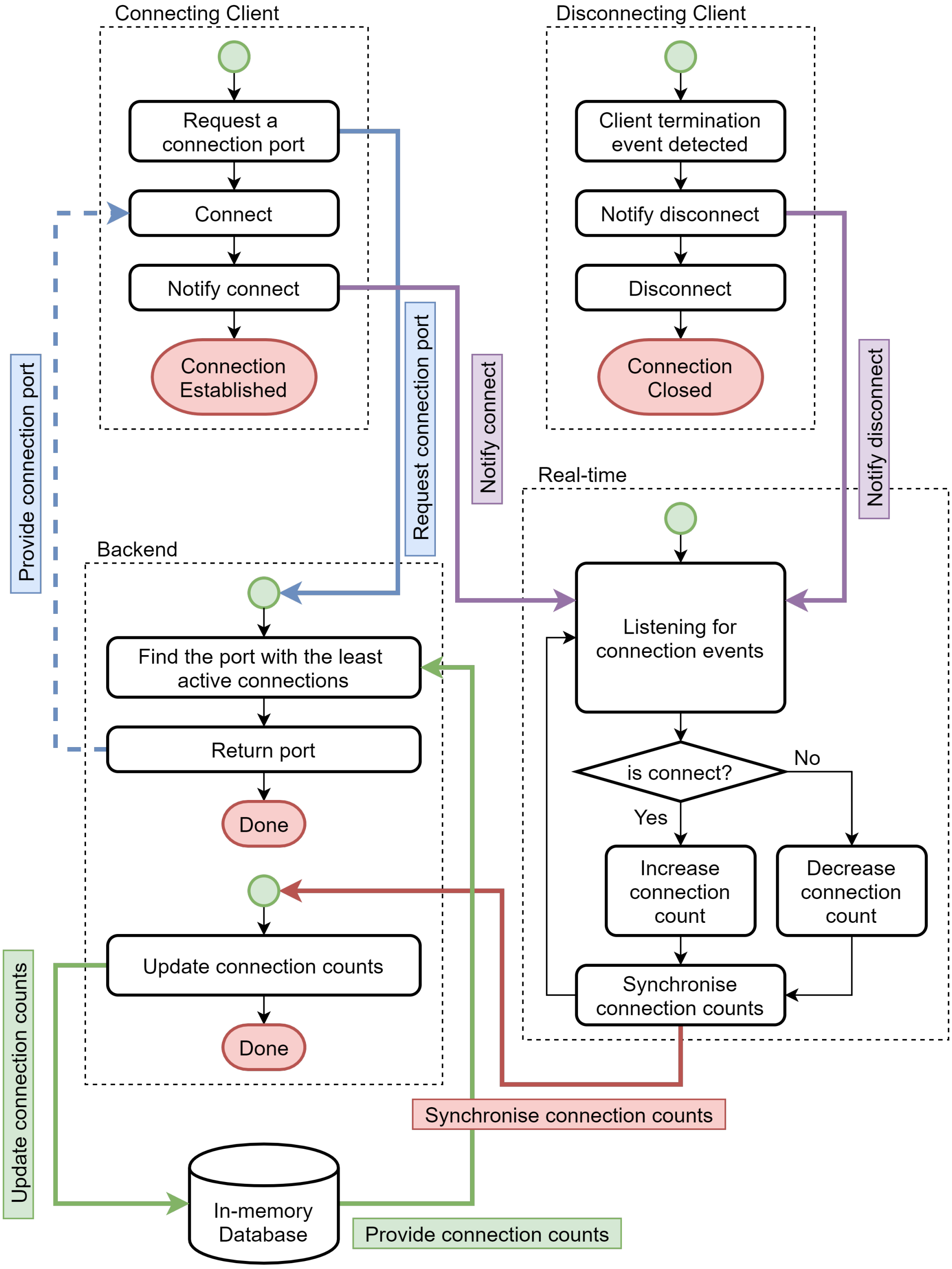

Figure 4.5: The context-aware load balancing process.

and broadcast the new data to the relevant clients. The client takes the new data and add it to the visualisation.

The benefit of using the real-time server is it significantly reduces the amount of backend access. Without the real-time server, the client have to periodically querying the backend regardless if there are new data or not. This is because there is no way to tell if a new piece of data is available at any given time. Assuming the querying rate is one query per second and there is only one client querying the backend. The periodic querying method would generate 60 queries per minute. By contrast, the real-time method only generates one and only one query, which is used for preloading the data. The load for the real-time method is shifted from backend to the real-time server and the number of messages from the real-time server is the number of times the microcontroller records the data. That means the worst case scenario is generating 60 data update messages per minute [6] to the client and the best case scenario is generating no data update as the microcontroller is not recording data. It is also worth noting that the cost of a query and data update is very different. A query involves accessing the on-disk database whereas the data update doesn't. Since accessing on-disk database is very slow by contrast, 60 real-time updates is more efficient than 60 queries.

4.3.6 Issuing command

The process of issuing a command involves the user issuing a command through the user interface and queuing the command in the database for the microcontroller to retrieve and execute. The process is shown in the figure 4.7. The process begins when the client sends the command to the backend, and then the backend enqueue the command inside the on-disk database. The order of the commands stored in the database must be maintained to ensure the correctness of the result after the

[6] The microcontroller records one sensing data every second.

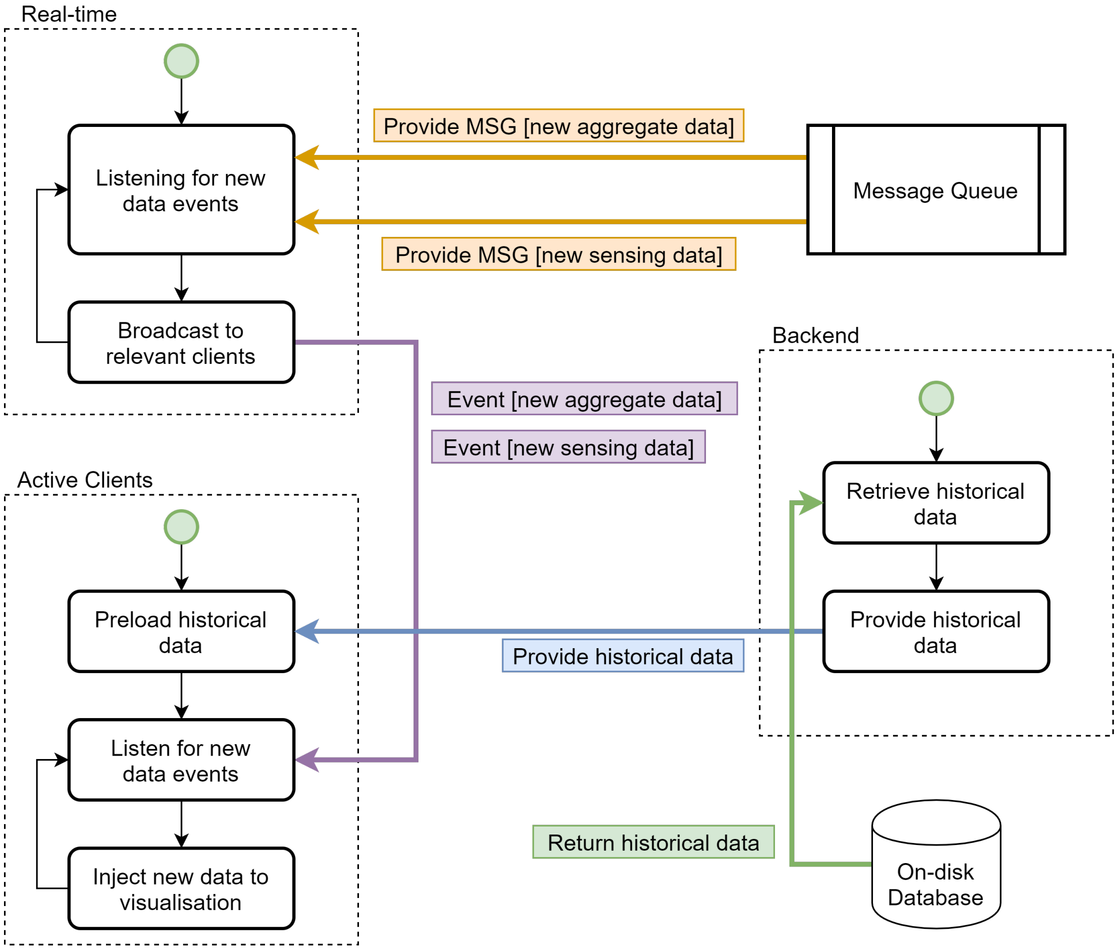

Figure 4.6: The data monitoring process.

microcontroller executes a series of commands.

4.4 Low-level architecture

In this section, the composite components shown in the top-level architecture are described in detail.

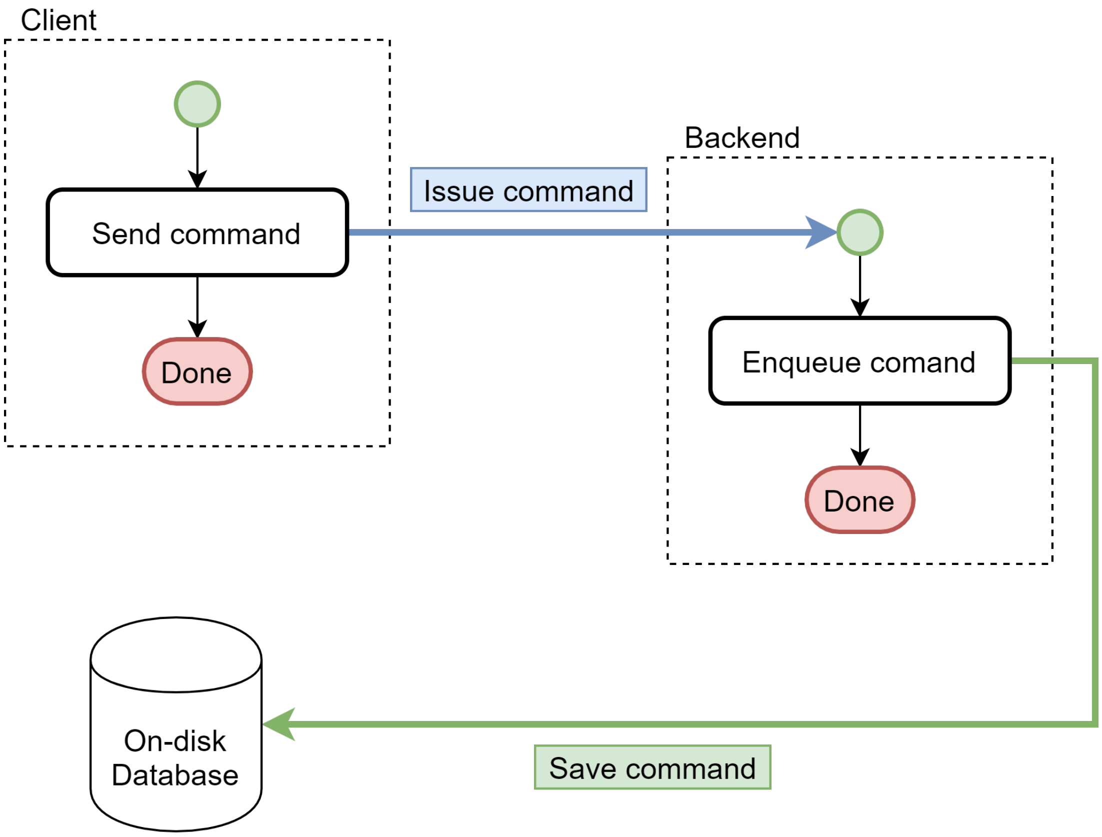

Figure 4.7: The process of issuing a command.

4.4.1 Remote sensing and simmulation

The figure 4.8 shows the remote sensing composite component is made up of the following low-level components in the experimental setup:

- Solar Panel
- Sensor
- Converter
- Battery
- Microcontroller

- LTE Modem
- Antenna

In the experimental setup, the solar panel generates electricity which is measured by the input sensor and then fed into the converter. After converting the electricity to the desired voltage, it is measured by the output sensor and stored in the battery. Then, the microcontroller reads the analog output from both sensors, computes the current and voltage readings, and pushes new sensing data to the backend through the LTE Modem using HTTP protocols. Additionally, if the commands are received, the microcontroller would send electrical signals to the converter to alter its behaviour.

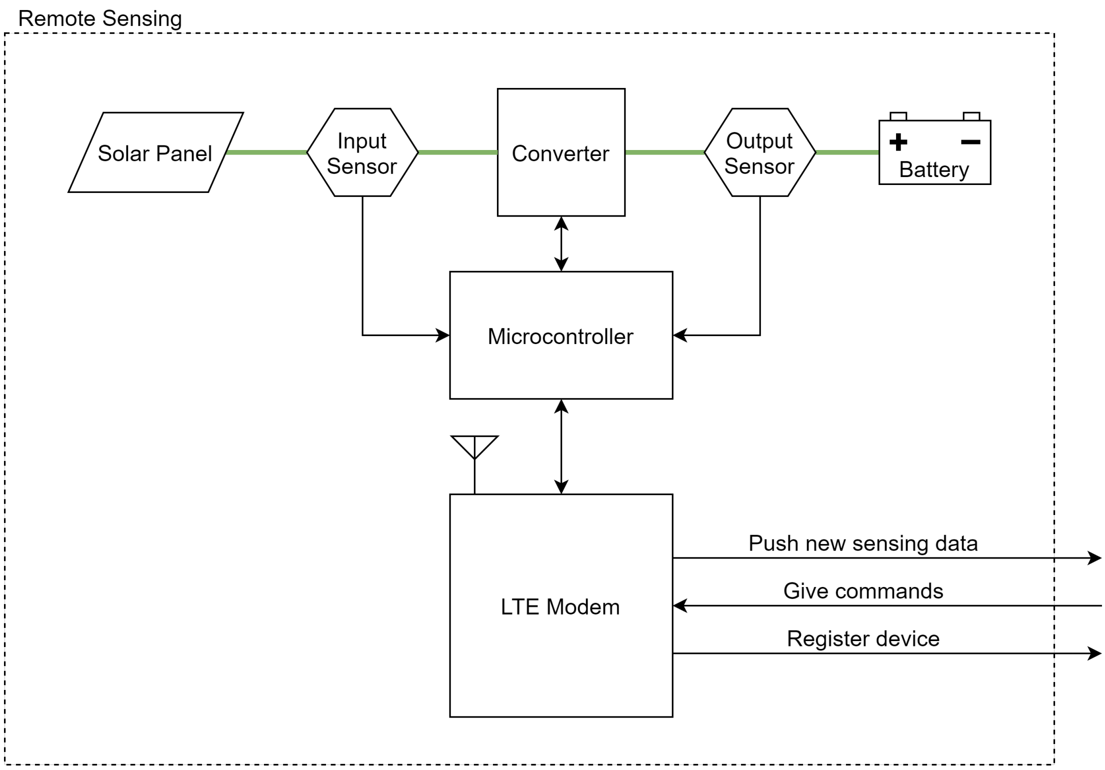

Figure 4.8: The architecture of the remote sensing composite component.

The low-level components can be varried greatly depending on the use cases, as long as it confronts to the behaviours that the proposed system expects:

- Registering with the backend.
  - Providing its device ID to the backend on startup, or request a device ID if it doesn't have one.
  - Remembering its assigned device ID given by the backend.
- Sending sensing data to the backend.
  - Current from the power generation device. (Input current)
  - Voltage from the power generation device. (Input voltage)
  - Current from the power converter. (Output current)
  - Voltage from the power converter. (Output voltage)
- Executing commands.
- Communicating with the backend over HTTP protocols.

A simulator was developed which confronts to the expected behaviours of the proposed system to generate realistic server load for benchmarking. The simulator has the capability to spawn hundreds or thousands of computer processes and each simulating a remote sensing component that is recording data and pushing them to the backend.

4.4.2 Backend

The backend composite component is shown in the figure 4.9 and consisting of the following components:

- Load Balancer

- Backend Instances
- Job Queue
- Async Worker

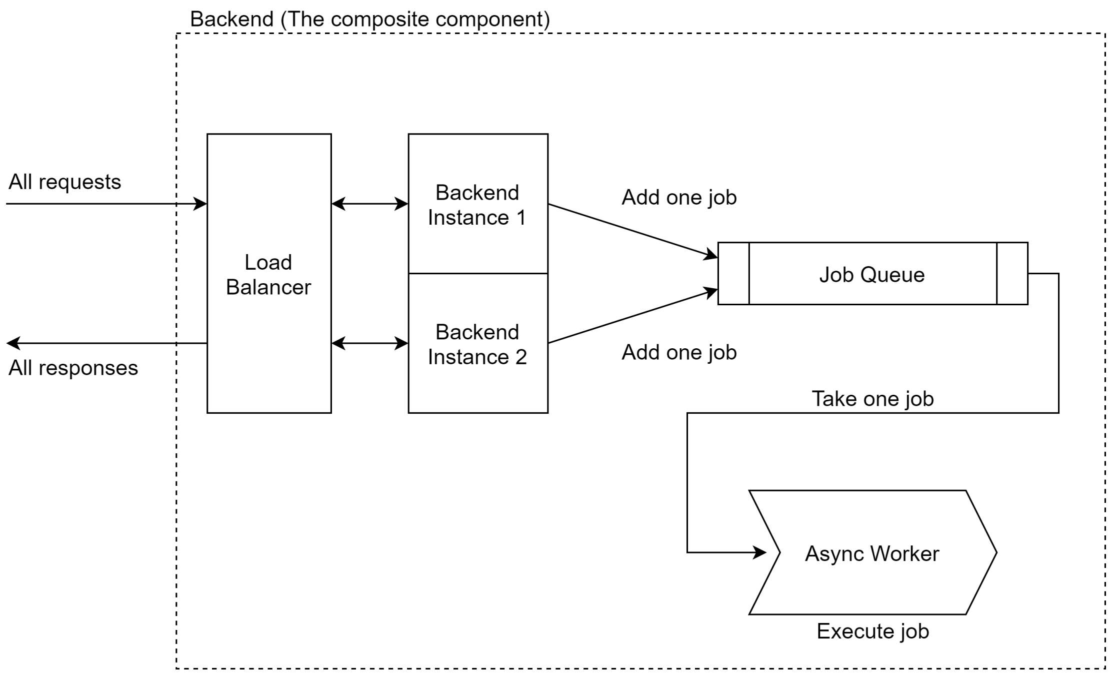

Figure 4.9: The architecture of the backend composite component.

All requests to the backend must go through the load balancer, which distributes the requests evenly to the two backend instances that are actually handling the requests and generating the responses. Among those requests, some of them are time insensitive[7]. This type of tasks are added to the job queue which allows them to be processed later. Finally, the jobs in the job queue are taken and executed by the async worker in a first in first out manner.

To understand how this processing mechanism allows more requests to be processed, we need to understand a common metric to measure the performance of

[7] Tasks that we don't have to wait for the result to be availabe before we proceed.

a server, the response time. The response time is the time from sending the request to receiving the response. Therefore, the response time can be broken into transmission time and processing time. The transmission time is the time it takes for a request to travel from the sender to the backend and the time it takes for a response to travel from the backend to the sender. The processing time is the time it takes for a request to be processed at the backend. Furthermore, the tasks are either time sensitive[8] or insensitive, we can break the processing time into the time it takes to execute time sensitive and insensitive tasks. If the average response time of a request can be reduced, then the backend can process more requests given the same amount of time.

The figure 4.10 illustrates the async worker accelarates the response time of a request, which allows the backend to process more requests. Assuming the transmission time, the processing time of time sensitive tasks, and the processing time of time insensitive tasks are identical with and without async worker. Without the async worker, tasks must be executed one after another. Therefore the processing time includes the time to execute time sensitive and insensitive tasks. With the async worker, we can add the time insensitive tasks into the job queue, which allows them to be executed at a later time. This is fine because the result of the time insensitive tasks are not important to processing the requests. Therefore, the processing time includes the time to execute time sensitive tasks and the time to enqueue a task to a job queue. There is no need to wait for the time insensitive tasks to finish and the time to enqueue a task is usually much faster, which allows the request to be processed faster, reduces the response time, and improving the performance of the backend.

[8] Tasks that we must wait for the result to be availabe before we proceed.

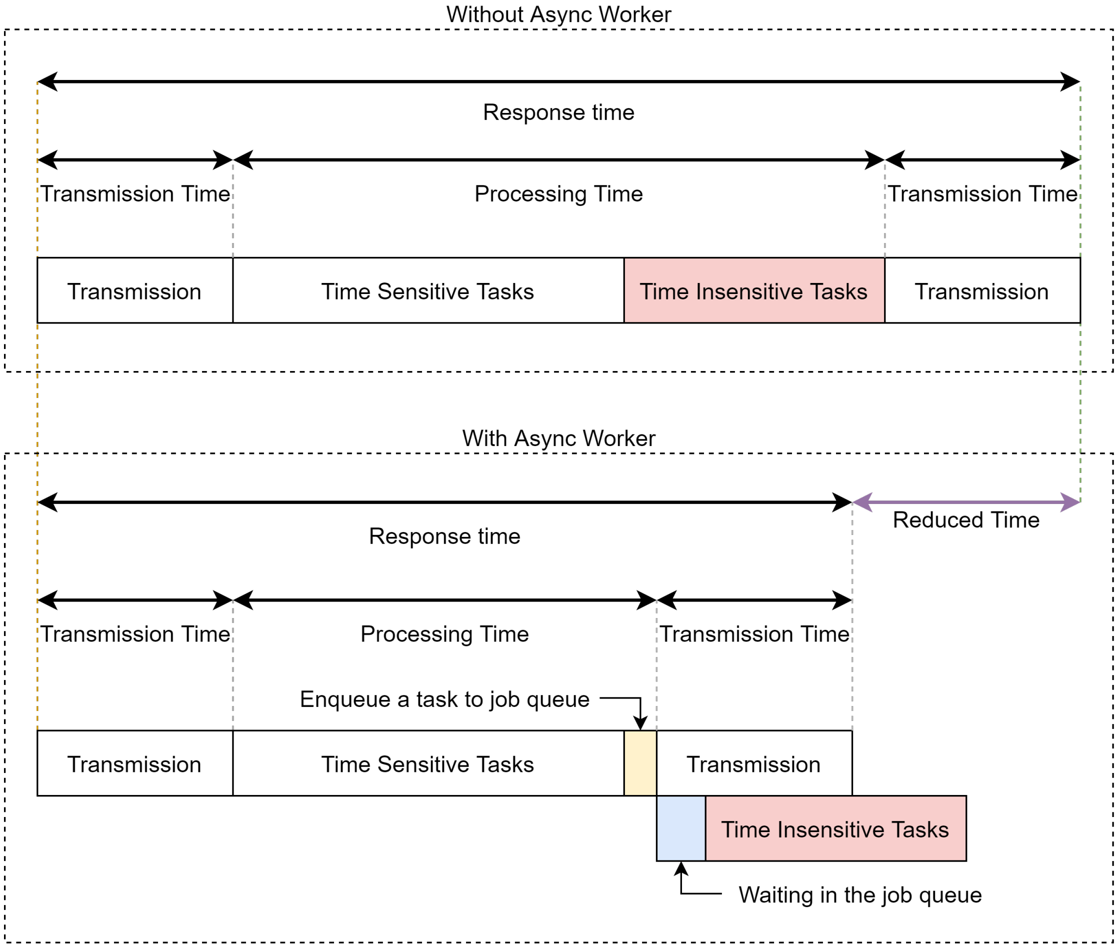

Figure 4.10: The comparison between using async worker and not.

4.4.3 Frontend

The frontend composite component is shown in the figure 4.11 and consisting of the following components:

- Load Balancer
- Static Web Content

Similar to the backend, all requests must go through the load balancer, and each thread in the load balancer is responsible for retrieving the static web content[9]

[9] Parts of the website that doesn't change.

and provide it to the client.

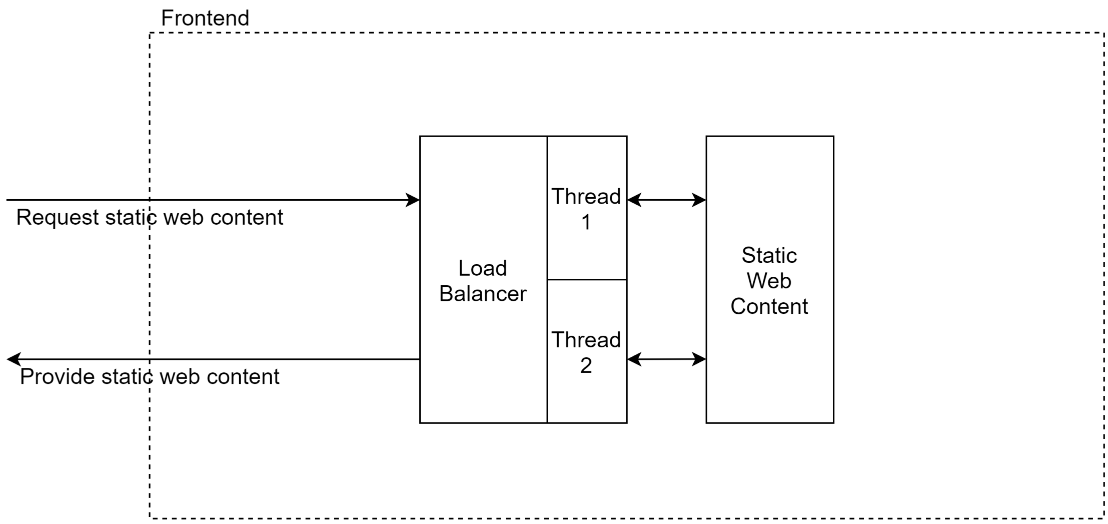

Figure 4.11: The architecture of the frontend composite component.

4.4.4 Real-time

The real-time composite component is shown in the figure 4.12 and consisting of the following components:

- Data Distributor
- Socket Instances

The data distributor receives data from external sources and distribute (emit) the data to the socket instances. Then, the socket instances send the data to their connected clients in the form of events.

To make it more efficient, the data distributor distributes the data to the socket instance if and only if the socket instance needs it. This is done by keeping track of what data that each socket instance needs. For example, in the scenario which is shown in the figure 4.13, when a client is connected to the socket instance 1,

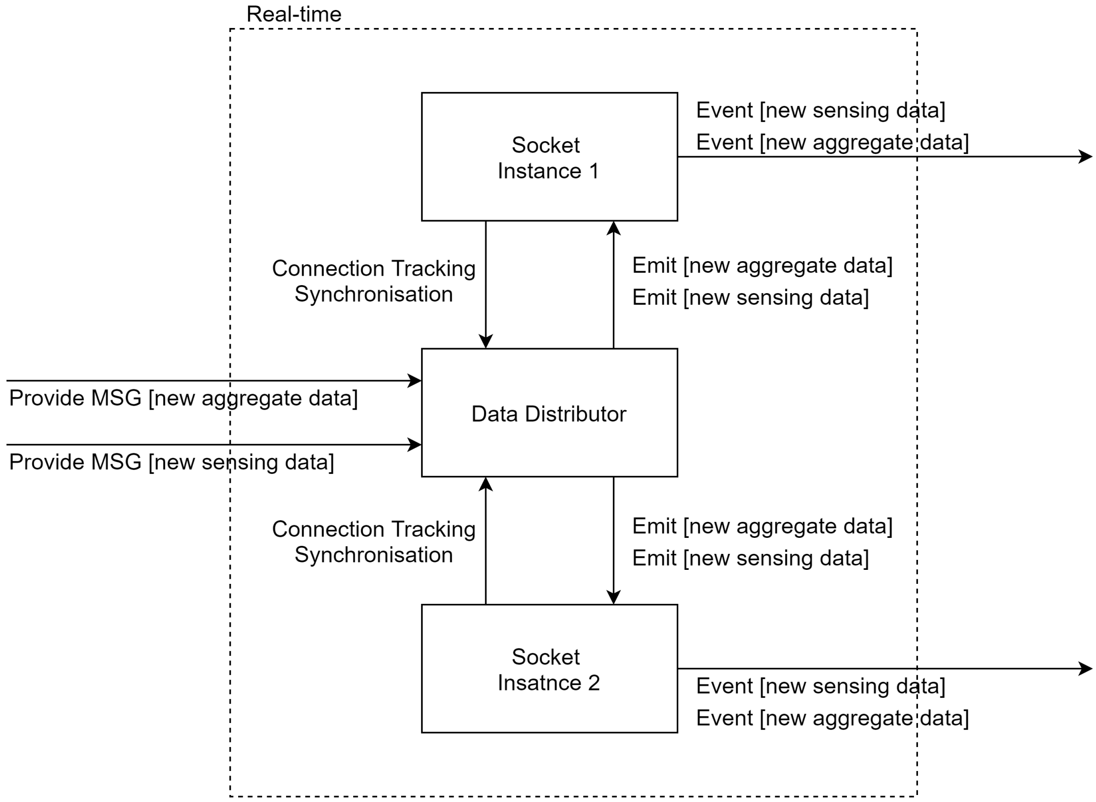

Figure 4.12: The architecture of the real-time composite component.

the client says it wants to receive data generated by the solar panel A. Then, the socket instance 1 notifies the data distributor through a connection tracking synchronisation request. The synchronisation reqeust tells the data distributor that the client on the socket instance 1 wants to receive sensing data from the solar panel A when it is available. Therefore, the data distributor knows if the socket needs a certain piece of data and send it to the socket.

If the client is disconnected, the socket would send another connection tracking synchronisation request to the data distributor and notifying this client no longer needs the data. This scenario is shown in the figure 4.14.

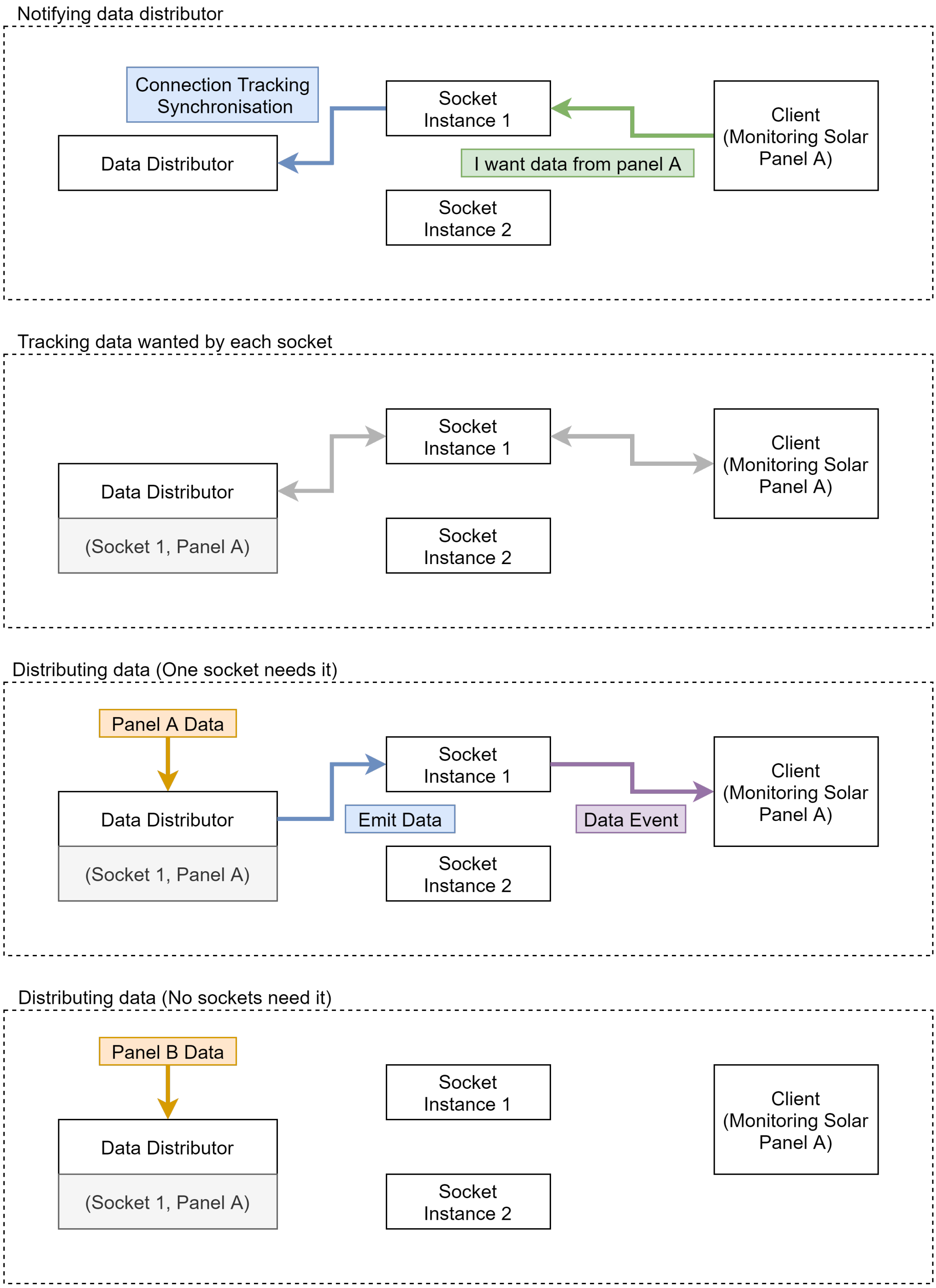

Figure 4.13: The data distributor innerworkings after clients connected.

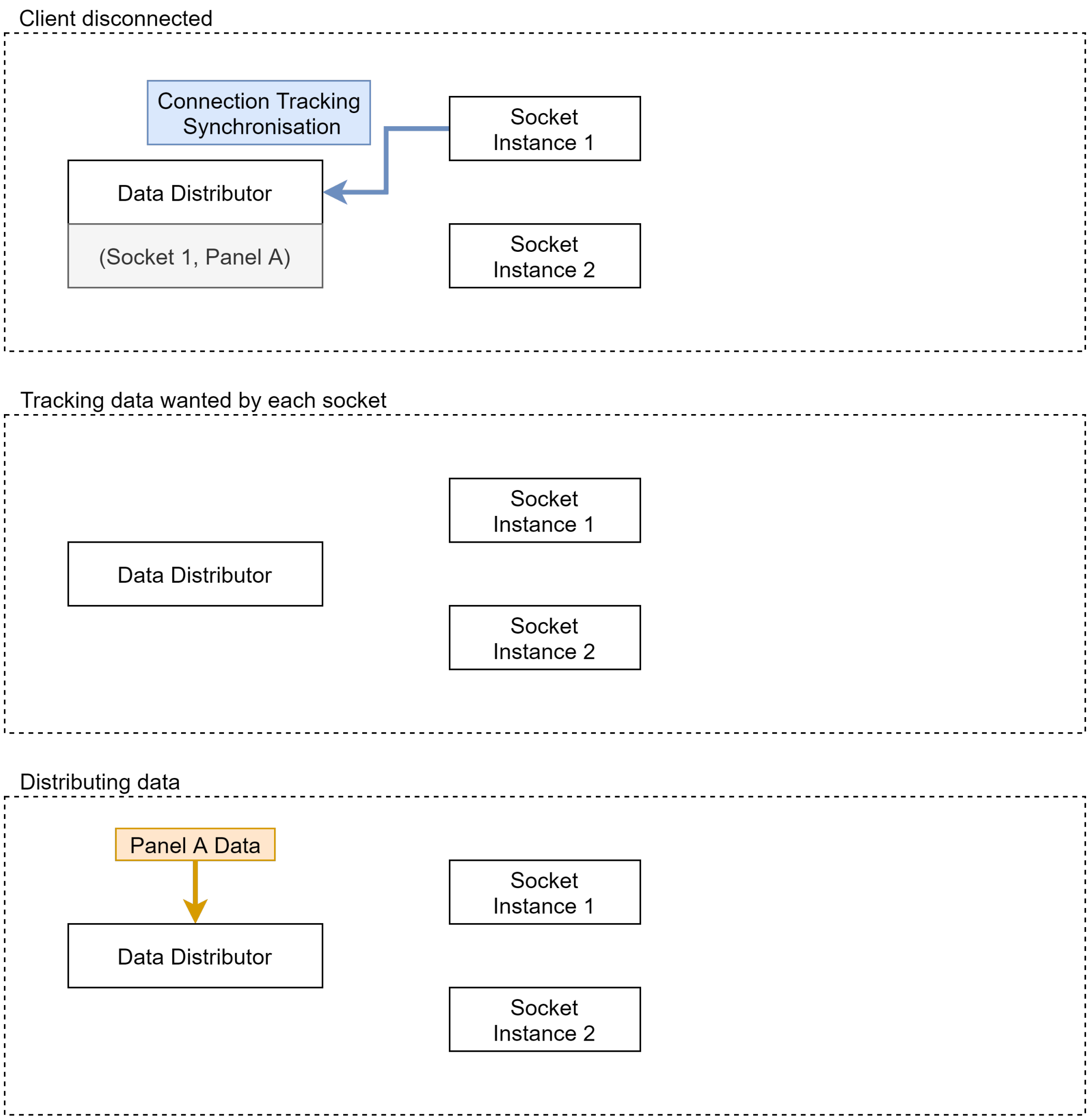

Figure 4.14: The data distributor innerworkings after clients disconnected.

Chapter 5

Benchmark

5.1 Aim

The aim for benchmarking is understanding the performance characteristics of the proposed system under various loads and system configurations.

5.2 Setup

The hardware used for benchmarking is detailed in subsection 3.4.1 and the critical technical specifications that affect the performance of the system are:

- 8 Core 16 Threads
- 4.3 Ghz Clock Speed
- 32 GB Memory
- Hardware Virtualisation Enabled

To observe the performance characteristics of the proposed system with horizontal scalling, additional hardwares used for horizontal scalling benchmarking are listed below:

- MacBook Pro 16 2019
  - 8 Core 16 Threads
- HP Envy dv6 2013
  - 4 Core 8 Threads

The software used for benchmarking is detailed in section 3.5 and the critical tehcnical specifications are:

- Ubuntu 20.04
- Docker Engine Enabled

Additionally, the proposed system have two configrations that can be tweaked to show how the performance can be impacted:

- Number of backend instances
- Number of async workers

The components of the proposed system are deployed to the benchmarking server using Docker Containers[1] to manage the runtime dependencies. Note that this method of deployment have minimal impact of the performance of system.

5.3 Procedures

5.3.1 Limited Computational Resources

The configurations listed in the table 5.1 are used to observe the behaviours of the proposed system with limited computation resources that is comparable to a modern laptop.

[1] A container virtualisation method, which allows the components to be run in an isolated environment that is always identical regardless of the environments of the server.

Table 5.1: A set of system configurations used for performance benchmarking with limited computational resources.

No. Backend Instances	No. Async Workers	No. Devices
4	4	50
4	4	100
4	4	200

5.3.2 Reasonable Computational Resources

The configurations listed in the table 5.2 are used to observe the behaviours of the proposed system with reasonable computation resources that is comparable to a high performance consumer grade desktop computer system.

Table 5.2: A set of system configurations used for performance benchmarking with reasonable computational resources.

No. Backend Instances	No. Async Workers	No. Devices
4	12	50
4	12	100
4	12	200

5.3.3 Horizontal Scalling

The proposed system is designed to scale, that is, having the capability of running multiple instances on multiple servers to distribute the load. The table 5.4 shows the configurations used for benchmarking where there are three physically separated servers contributing to executing the tasks at the same time. The computation resources provided by the servers are shown in the table 5.3

Table 5.3: Contribution to computational resources.

Server	Threads	No. Backend	No. Workers
Benchmarking Server	16	4	12
MacBook Pro 16 2019	16	0	16
HP Envy dv6 2013	8	0	8

Table 5.4: A set of system configurations used for scalability benchmarking.

No. Backend Instances	No. Async Workers	No. Devices
4	12 + 8 + 16	100
4	12 + 8 + 16	200
4	12 + 8 + 16	500
4	12 + 8 + 16	1000

5.3.4 Generating Loads

The simulator is used to generate realisitic loads by spawning computer processes that are accssing the backend in the same manner as a real remote sensing device. For each simulated remote sensing device, it sends the sensing data to the backend every second for a total of 100 iterations to create a sustained load.

5.3.5 Measuring Response Time

To quantify the performance, the response time of each request is measured and recorded. Once a simulated device has finished sending data, the response time for each iteration is saved to a CSV (Comma Seperated Values) file. Once all simulated devices have finished sending data, the average response time for each iteration is calculated and saved to another CSV file.

5.4 Analysis

Chapter 6

Real World Testing

6.1 Setup

6.2 Procedures

6.3 Analysis

Chapter 7

Conclusion

7.1 Implications

7.2 Future work

Bibliography

[Nuv,] *Nuvoton M263A*. Nuvoton.

[Que,] *Quectel EC25*. Quectel.

[ets, 1995] (1995). Digital cellular telecommunications system (phase 2);alphabets and language-specific information (gsm 03.38).

[3gp, 2020] (2020). Gprs adds packet-switched functionality to gsm networks.

[Adhya et al., 2016] Adhya, S., Saha, D., Das, A., Jana, J., and Saha, H. (2016). An iot based smart solar photovoltaic remote monitoring and control unit. In *2016 2nd International Conference on Control, Instrumentation, Energy and Communication (CIEC)*, pages 432–436. IEEE.

[Shariff et al., 2015] Shariff, F., Rahim, N. A., and Hew, W. P. (2015). Zigbee-based data acquisition system for online monitoring of grid-connected photovoltaic system. *Expert Systems with Applications*, 42(3):1730 – 1742.

[Siregar and Soegiarto, 2014] Siregar, S. and Soegiarto, D. (2014). Solar panel and battery street light monitoring system using gsm wireless communication system. In *2014 2nd International Conference on Information and Communication Technology (ICoICT)*, pages 272–275. IEEE.

[Syafrudin et al., 2018] Syafrudin, M., Alfian, G., Fitriyani, N., and Rhee, J. (2018). Performance analysis of iot-based sensor, big data processing, and machine learning model for real-time monitoring system in automotive manufacturing. *Sensors*, 18(9):2946.