# SMARTDART - A cloud-based architecture of the DART tsunami warning system

MSc Research Project
Cloud Computing

Rihand Parde
Student ID: 23115165

School of Computing
National College of Ireland

Supervisor: Sai Emani

**National College of Ireland**
**Project Submission Sheet**
**School of Computing**

| | |
|---|---|
| **Student Name:** | Rihand Parde |
| **Student ID:** | 23115165 |
| **Programme:** | Cloud Computing |
| **Year:** | 2024 |
| **Module:** | MSc Research Project |
| **Supervisor:** | Sai Emani |
| **Submission Due Date:** | 12/12/2024 |
| **Project Title:** | SMARTDART - A cloud-based architecture of the DART tsunami warning system |
| **Word Count:** | 7579 |
| **Page Count:** | 23 |

| | |
|---|---|
| **Signature:** | |
| **Date:** | 25th January 2025 |

# SMARTDART - A cloud-based architecture of the DART tsunami warning system

Rihand Parde
23115165

**Abstract**

The DART tsunami warning system has been in operation for over 20 years with limited research done to improve it. It still operates on two decade old technology, which leaves a research gap to improve its functionalities using modern technologies. SMARTDART is a novel approach towards enhancing the capabilities of the DART tsunami warning system using cloud technology and fog and edge computing. This paper describes the working of the current system, highlights its limitations based on a detailed literature review of previous work, and proposes an AWS-based solution. A simulation project was developed in iFogSim consisting of two models - DART and SMARTDART. DART simulated the current system as it originally functions while SMARTDART simulated the proposed AWS-based solution. Results showed that the SMARTDART model outperformed the DART model in terms of reduced latency and increased throughput. When practically implemented, this project is expected to reduce the latency of the current DART system, make it more fault tolerant, increase its scalability, and reduce the production and operational costs in the long term.

## 1 Introduction

Tsunamis are some of the most widely occurring natural disasters in the world. Although major tsunamis do not happen frequently, they do however cause immense destruction, leading to massive losses of lives and property. To safeguard people from tsunamis, scientists employ various technologies, ranging from using infrared to detect potential tsunami waves to installing underwater sensors that detect unusual changes in water pressure and temperature which could lead them to building up into a tsunami. These devices are called tsunameters, with DART, short for Deep-ocean Assessment and Reporting of Tsunamis Gonzalez et al. (1998), employing a high number of tsunameters around the globe to detect these anomalies in the waters. The working of the DART system is simple and straightforward - bottom pressure recorders (BPR) are sensors, placed on the sea floor, that detect sudden changes in water pressure and temperature. These sensors wirelessly transmit the data to DART surface buoys, which in turn forward this data to tsunami warning centers via the Iridium Satellite Network, where this data is analyzed before a warning is issued nationally. The current system, though, still has not been upgraded from the 20 year old technology on which it still operates, which leaves an important research gap for the potential to improve it using modern technologies and infrastructures. Thus, the research in this paper proposes one such solution, which is to implement cloud technology and fog and edge computing to this system to boost its operative performance.

The research question of this paper can be summarized in the following way:

**How can the cloud-based SMARTDART system ensure long-term fault tolerance, scalability and cost-effectiveness compared to the existing DART system?**

The following are the three major objectives of this research:
1. To present an AWS cloud-based architecture that will enhance the capabilities of the DART tsunami warning system
2. To improve the fault tolerance of the current DART system through fog and edge technology
3. To demonstrate the cost-effectiveness of the proposed cloud-based solution compared to the current system

So far, most of the research done on this topic had been in the form of infrastructure and algorithm development. The DART generation 4 buoys were upgraded with the latest hardware and its bottom pressure sensors installed with optimized algorithms for better accuracy. When it comes to the potential of cloud technology to significantly enhance this system, no such research has been done for this particular system. It is for this reason and the shortcomings of the current DART system described in the next section, that this study proposes SMARTDART - a cloud-based fog and edge technology model of the DART tsunami warning system. The structure of this research paper is as follows:
1. Section 1 presents an introduction to the topic, describing a brief overview and highlighting the research question and its objectives.
2. Section 2 consists of a detailed literature review. This section contains the disadvantages of the current DART system and describes all the research done on this particular system and research that were conducted on similar systems.
3. Section 3 describes the procedures and techniques that are used to conduct the experiments to validate the research.
4. Section 4 presents and describes the architecture of the cloud-based SMARTDART tsunami warning system.
5. Section 5 describes the experiments performed to produce an outcome which will prove why the proposed solution is better than the current system.
6. Section 6 evaluates the results of the experiment and presents a detailed analysis of the outcomes.

Section 6 concludes the results of the research with a brief consideration for the future direction of this research.

## 2 Related Work

Figure 1 shows the visual diagram of how this system works.

From a higher perspective, it looks like a simple system that functions as required. However, the current DART system has a number of issues that are open to research:
1. Although DART buoys are designed to be durable enough to handle daily rough situations, there are times when they malfunction due to not being maintained timely

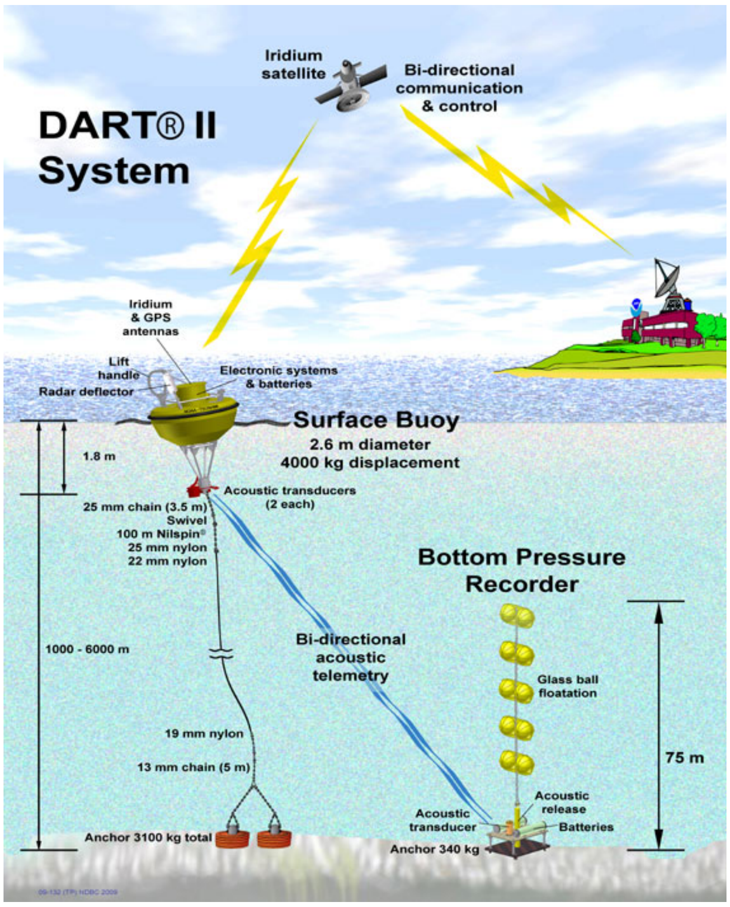


Figure 1: The working of DART tsunami warning system (source: `https://www.ndbc.noaa.gov/images/dart/dart_mooring.jpg`)

or due to the ocean environment itself. This causes some buoys to become inoperative, which means they cannot send vital data to the tsunami centers. On a a network spanning worldwide, this could be catastrophic as mentioned in a paper by Lawson et al. (2012) which describes how the 2001 Boxing day tsunami, 2010 Chile tsunami, and 2011 Honshu tsunami happened due to unreliable and inoperative tsunami warning systems.
2. High latency, which is caused by data transmission over long distances, is an issue that needs to be acknowledged. First, is the transmission from the sensors to the buoys, then from the buoys to the satellite network and finally to the tsunami warning centers, where the data is processed. Although the current system functions as required, latency could be further reduced by bringing the processing power closer to the edge of the network, which will be described in more detail in further sections.
3. The current DART network is costly to set up and operate. Improvements were made in the 4th generation DART buoys which greatly reduced its deployment costs but setting up tsunami warning centers still adds to the infrastructure costs, not to mention the buoys themselves are costly. These centers are important for analyzing tsunami data but in the modern world which offers far more advanced technology, setting up separate physical structures just to perform basic analytical tasks is unnecessary when better options are available.
These are the three major issues that are open for research in this niche topic. The existing system functions as needed for now but plans are in motion for upgrading it

since it has been running on the same technology it was developed with 20 years ago. This leaves a major research gap which can be filled with the use of cloud technology to further enhance the DART system's capabilities while offering cost effective solutions.

## 2.1 Tsunami Warning Systems

A majority of the technological advancements in tsunami warning systems have been achieved in the form of hardware improvements. Angove et al. (2015) examines the capabilities of the current tsunami warning system in the US, ranging from their detection and warning strength, their mitigation capabilities to the technological advancements made by the National Oceanic and Atmospheric Administration (NOAA), and the potential future directions for further research. The paper puts more emphasis on the DART's buoy network that spans across the oceans. Although many of the improvements mentioned in this paper are hardware-based, such as the upgraded generation 4 DART buoys and undersea fiber optic cables, it also highlights how financially and logistically challenging it is to maintain the DART network, but the paper fails to elaborate on this issue. This issue could be addressed with AWS's IoT core dashboard that will keep track of all the fog devices (buoys), a part of the proposed solution.

Percival et al. (2016) demonstrated how tsunami wave forecasts degrade when one or more DART buoys become inoperative based on the results gathered from a simulated environment. The results showed that, out of 14 buoys, even when one or two buoys become inoperative, the RMSE ratio becomes greater than 4, resulting in a 'red' status, which means a serious deterioration of the network. This article has developed a performance measure to simulate a particular situation but it does not address the core of the issue - which is the inability of the buoys to communicate with each other to keep tab of their statuses. With the help of smart sensors, the buoys can intercommunicate to keep check on their health statuses and AWS's analytics will send a timely alert if an inoperative buoy is found.

A web-based tool, called INSPIRE, which stands for Internet based Simulation Platform for Inundation and Risk Evaluation, was developed by Srivihok et al. (2014) that estimates inundation and loss based on user-entered parameters. The paper describes the development and working of this tool, with it producing mixed results in terms of accuracy, which could have been made better it AWS and its analytical services to mitigate the loss caused by tsunamis.

Described by Taft et al. (2009) is a standard buoy prototype and the results of its initial testing. This prototype was developed by the National Data Buoy Center (NDBC) and the tests were deemed successful with the buoy being easy to service and sustainable. The greatest achievement of this research, however, was in the form of cost savings due to this buoy's capability to be serviced at the sea itself which saved transport costs by a major margin. The only shortcoming of this paper is that it does not explore the fault tolerance capabilities of a network consisting of the new buoys. The costs could be further reduced by hosting the network on AWS and using affordable yet durable smart sensors on the buoys.

Wang et al. (2023) explored a new method of tsunami forecasting using a near-field forecast method called tFISH in a planned Peruvian DART buoy network. Amongst four discoveries produced by their experiments, one stood out in the context of this paper's research - forecast accuracy generally improved when they increased the number of DART buoys although for that particular study, the increase was not significant. This shows the effect the number of active buoys have on tsunami forecast, and if even one were to fail and go undetected, the system's quality reduces by a significant margin.

The article by McClure et al. (2023) describes the history of research and development in the field of tsunami warning technology at NOAA's Pacific Marine Environmental Laboratory (PMEL) over the last 50 years, beginning from 1965 all the way to 2023. The most striking part of this article, however, is the conclusion that states the future development of a near-field warning component that will make the data available in 10 minutes or less after the earthquake stops. The article ends with the mention of how DART-4G buoys are being developed to detect tsunamis closer to their source, with artificial intelligence (AI) being used in recent years to improve warning capabilities. However, fog and edge technology has the potential to process data closer to the edge of the network, which is the gap that is being explored in this article's future work.

Adiprabowo et al. (2024) makes detailed comparisons between high frequency (HF) radar and DART radar, ranging from their real-time data transmission capabilities to their wave measurements. DART radar, in particular, has been found to detect a tsunami about an hour before it reaches the coast, making its detection quicker than HF radar's. DART's network is also wider, spanning upwards of 500 kilometers from the coast. The disadvantages of DART radar are the high costs of installation and operation, their limited numbers, and their vulnerability to disruption due to earthquake damage. The paper mentions how IoT can be integrated with HF radar for faster real-time data processing but judging by the advantages the DART radar has over the HF radar, an IoT architecture would suit it better, something that this paper does not consider.

## 2.2 Cloud computing in disaster management

A cloud and IoT-based flood risk mitigation system has been developed by Siek and Larry (2021) whose functioning is similar to that of the SMARTDART's. The sensor data from physical Arduino devices is processed by AWS IoT and stored in DynamoDB table with MQTT being providing the communication protocol. A basic web application was developed for data visualization. The sensors monitor water levels and once they reach a certain height, an alert is sent to authorities. Although this system has been successfully tested, it does not mention how it compares in terms of cost and maintenance compared to current flood risk mitigation systems. This issue could be addressed by doing a thorough cost analysis using AWS's service documentations.

Qiu et al. (2014) have proposed a cloud-based emergency management system, called Smart Cloud Evacuation System (SCES). This system is divided in two parts - the front-end and the back-end. The front-end which consists of a large number of sensors placed in disaster-prone areas that collects data based on damage to the buildings. The back-end consists of multiple cloud datacenters that sends alerts to authorities depending on the analysis of the sensor data from the front-end. A unique feature of this system is its innovative social-media analysis feature which analyzes the social situation through the news from that particular disaster site from social media applications and helps the system to generate emergency response accordingly. The only shortcoming of this paper is that it does not specifically mention which cloud infrastructure it uses. For this purpose, AWS would be an ideal cloud infrastructure since it provides high performance analytical services, high scalability on its serverless infrastructures, and better fault tolerance compared to other cloud vendors.

A study by Abdelaziz et al. (2024) reviews existing cloud solutions for their disaster recovery (DR) capabilities during situations where natural disasters strike areas where the cloud datacenters are located. Data was collected from three major cloud service providers using IoT sensors - AWS, GCP and Microsoft Azure. Two distinct models are proposed in this paper - a grid decision model and an ANN model, for automating DR site selection. The backup locations have been compared based on various metrics, which includes their location, redundancy, security measures, network connectivity and RTO and RPO, to name a few. The paper concludes that the three most important factors for choosing a backup site were redundancy, proximity, and datacenter location. Although the paper mentions in its abstract that their solution is cost effective, it does not elaborate on this cost-effectiveness in the rest of its content. The information provided in this paper proved why using a cloud-based infrastructure provides better fault-tolerance than the current DART network.

Described in the survey paper by Ujjwal et al. (2019) are the major open challenges that arise when adopting cloud computing for natural hazard modeling and management. The paper details the development of the concept of a natural hazard model as a deployable service. Some of the issues these models face are high power requirements, high data requirements, concurrent access, time-critical requirements and downtimes due to natural disasters. The paper concludes by stating that the huge reliance on the internet can cause major issues due to the possibility of it breaking down during disasters, leading to the failure of the cloud infrastructure. The solution they proposed was the integration with IoT and fog and edge networks, which can continue to provide disaster-related services even during network failures, acting as transitional data relays. If an AWS infrastructure is configured the right way, data could still be transmitted in the natural hazard model even when it is offline, a feature that is present in the proposed solution, which means no more heavy reliance on the internet.

Krichen et al. (2024) wrote a survey paper on the use of various technologies, such as satellite imaging, remote sensing, radar, IoT, smartphones and social media, for the management of natural disasters, with the goal of predicting, responding and recovering from those disasters more effectively. The survey identifies limitations and challenges that come with these technologies. In context of IoT, this paper describes how it is efficient for natural disaster management due to the presence of low cost sensors and real-time data processing capabilities. The only limitations highlighted about IoT systems are security concerns and data privacy. Also, due the continuous advancements in technology, it is difficult for emergency responders to keep up with them. The survey paper concludes with the consideration of using AI to swiftly detect disasters before they happen and blockchain networks to strengthen privacy and security. In regards to the proposed solution, while the upfront expenses to upgrade the system to an AWS-based architecture might be high, the long term costs associated will be significantly lower than what is incurred by the current DART system.

## 2.3 IoT for disaster management

The experiments conducted by Kusuma et al. (2022) shows how latency is affected with increase in quality of service (QoS) levels of MQTT. The experiments were based on a simplified virtual testbed model of Indonesia Cable-Based Tsunameter (INA-CBT) and displays the MQTT message transmission power over the land station (LS) and read down station (RDS) communication segments. The results showed that, since the bottom pressure recorder (BRP) size and transmission frequency are small, the latencies are also shorter for them. However, increasing the QoS also led to higher latency. The experiments in this paper, however, should have also considered how latency is improved during data processing at the edge of the network as present in SMARTDART's architecture.

Živić et al. (2023) makes a detailed comparative analysis of three widely used IoT protocols - MQTT, CoAP and ZeroMQ. The experiment involved setting up a controlled IoT testing environment consisting of three IoT devices, each configured to utilize the respective protocols. CoAP protocol was found to be more efficient for on-demand services, ZeroMQ was ideal for peer-to-peer messaging, and MQTT was best suited for low-latency data transportation, which is ideal for real-time data transmission scenarios such as the SMARTDART network.

A detailed article by Esposito et al. (2022) reviews a large volume of papers based on the implementation of IoT solutions for the development of Early Warning (EW) systems, namely for floods, earthquakes, tsunamis and landslides. In context of tsunamis, the literature review yielded an interesting result - one was that satellite communication remains the most reliable method of data transmission from the buoys to the warning centers, however, when an IoT solution is implemented, then LoRa and GSM proved to be better choices due to their low-powered long range data transmission capabilities. The article discusses the potential of IoT solutions to provide timely warnings, economical benefits and reduced latency due to resources being closer to the end devices in the network. Also mentioned in the article is a research gap, which is the lack of fault tolerance in detecting failed nodes in the network. In context of the DART system, these would be inoperative buoys. Though the article mentions the possibility of integrating cloud solutions with IoT, it does not consider this solution in the section detailing EW systems for tsunamis.

To conclude this section, the main issues with the current DART tsunami warning system are service degradation due to the unawareness of inoperative buoys, high latency caused by long distance data transmissions, and high costs incurred by setting up on-site infrastructure for data processing. The proposed solution, called SMARTDART, can solve these issues by allowing the buoys to communicate with each other through smart sensors, processing data at the edge of the network which will greatly reduce latency, and using pre-built AWS infrastructure, thereby saving the costs of setup, operation and maintenance of an infrastructure.

# 3 Methodology

The proposed SMARTDART system will be an upgrade of the previous DART system, being more fault tolerant, faster, and more cost effective than its predecessor. To demonstrate the practicality of this solution, a simulation project was developed in iFogSim and its output transmitted to AWS through for processing. An alert is generated as an SMS based on when the output exceeds the pre-defined threshold levels. The detailed overview of the tools and technologies used for developing this project are as follows:

## 3.1 iFogSim

Access to physical hardware, such as real bottom pressure recorders and DART buoys, was not possible due to their cost and transportation arrangement. For this reason, iFogSim was used for simulating a fog and edge environment in this project. iFogSim is a toolkit specially designed for developing fog and edge computing based simulations. It allows the development and simulation of physical, logical and management devices entirely using software Yousuf Khan and Rahim Soomro (2022). Users can set up virtual data centers, fog devices, edge nodes, provision resources to them such as RAM, power, operating systems, etc. This toolkit was chosen for the project due to its real-time data processing capabilities, the high accuracy of its system state, reliability and high scalability. Also, many classes, which include fog devices, tuples, sensors, actuators and applications, are built-in to the toolkit, saving time and resources of separately developing them. Java is the programming language with which the SMARTDART simulation was developed due to its great compatibility with the iFogSim toolkit.

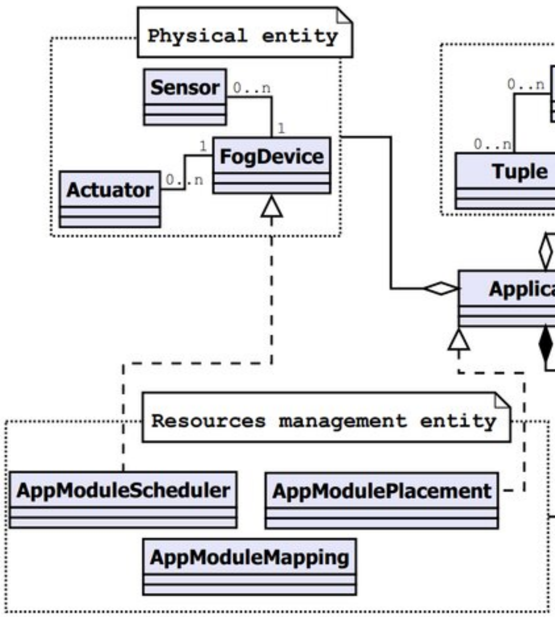


Figure 2: iFogSim components class diagram Naas et al. (2018)

## 3.2 Eclipse IDE

Two choices of integrated development environment (IDE) were available for building this project - Eclipse and IntelliJ. An in-dept comparison was made between these two and in the end, Eclipse was selected due to the following reasons:
1. Simple user interface (UI).
2. Consumers fewer computing resources, which is ideal for such a lightweight project.
3. High compatibility with iFogSim, since its official documentation recommends using it with Eclipse IDE Awaisi et al. (2021).
4. A large community support, which helped in troubleshooting a lot of the bugs and issues that arose while developing this project.

## 3.3 Amazon Web Services

For data processing tasks at the edge of the network, cloud technology provides a major advantage over traditional methods, which makes it an essential task to select the best cloud service provider. The three major cloud vendors in the market right now

are Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). Since this project is based on IoT, these three cloud vendors were compared specifically for their capability to provide support for IoT related applications. The results yielded by Muhammed and Ucuz (2020) proved that AWS would be most ideal for this project due to the following reasons:
1. Devices can be easily connected from an external source with simple integration, especially in simulation environments, through import of JAR files.
2. AWS provides more hubs compared to Azure and GCP, which, when implemented practically in the real-world, would provide a much more efficient device-to-device communication. This is essential for the SMARTDART buoys' fault tolerance capabilities.
3. AWS and GCP were found to have better security, but GCP falls short due to its other features being not as up to mark for IoT applications when compared to AWS.
AWS is best suited for this project due to its ease of use, large volume of hubs, and its security features.

## 3.4 Evaluation Parameters

The proposed solution should be quicker, fault tolerant, and cost efficient compared to the current DART system. The following are the parameters that will be evaluated in order to prove how the SMARTDART is better than what the current state-of-the-art system has to offer:

Latency: The measure of the total time it takes from data generation by the sensors to the alert generation through an SMS text. Latency will be measure in minutes. The current DART system has a latency of 3 minutes and this project aims to demonstrate a latency less than that of the current system.

Throughout: The frequency of data transmission per second. The current bottom pressure sensors transmit data to the surface buoys every 15 seconds. SMARTDART will demonstrate a higher throughput by being able to transmit data more frequently.

Cost Effectiveness: DART network is expensive to operate and maintain. Cloud integration has the potential to minimize those costs and this will be presented in more detail in the Analysis section.

The proposed solution's goal is to demonstrate how integration with cloud and fog and edge technology will enhance the DART tsunami warning system's capabilities, making it faster, more fault tolerant, and cost effective. Two simulation models will be developed in Eclipse IDE - one for DART and one for SMARTDART. Their outputs will be used for generating the metrics for comparison.

# 4 Design Specification

## 4.1 Simulation Architecture

The simulation model of SMARTDART consists of iFogSim code and AWS services. The code, written in Java, produces an output that randomly generates readings from the pressure and temperature sensors. Pre-defined threshold values are stored in a DynamoDB table. For example, pressure might have a threshold value of 1050 hPa and temperature might have a value of 40°C. Based on these, if the generated readings are under the threshold levels, there will be no action and the code will continue to execute. If the reading exceeds both the threshold values, then a Lambda function will be triggered. The function signals an SNS topic to send an SMS text to any device subscribed to a mobile network. In this case, the text will be sent to a smartphone with a valid SIM card, alerting the user of a potential tsunami hazard. The code will then continue to execute for a specified number of iterations. The entire process, in practice, happens instantaneously. Figure 3 shows the flowchart of the implementation of the simulation.

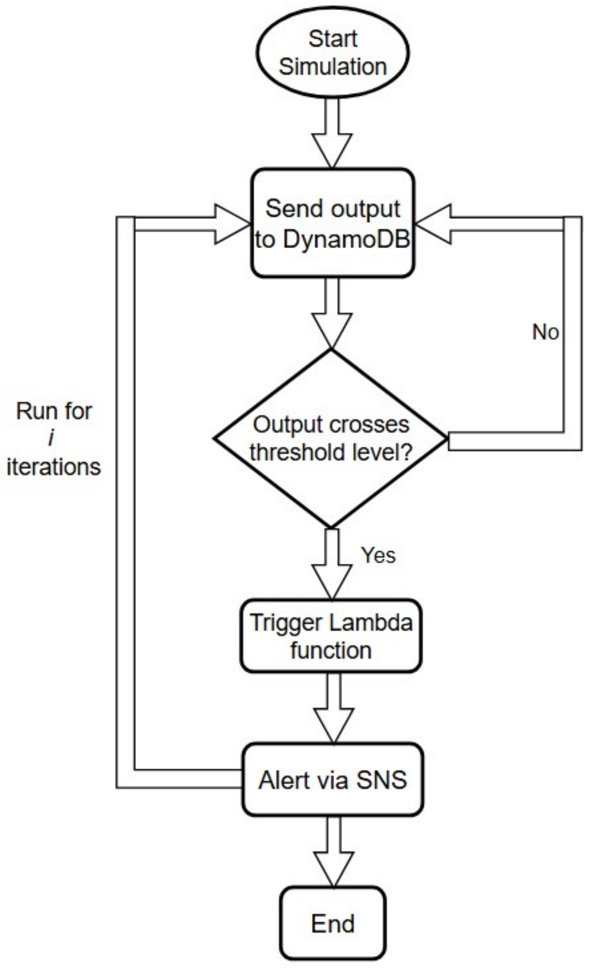


Figure 3: Flowchart of the simulation

## 4.2 Practical Architecture

The real-world implementation of this solution, called the SMARTDART, will be a little different from the simulation developed for this research but their fundamental concepts are the same. In practice, smart sensors will be placed on the ocean floor. These will be the edge devices which will collect raw data from their environment. The data will be transmitted as signals to the smart surface buoys, which will be the fog devices. This raw data will be processed at the buoys itself and get transmitted to the Iridium satellite network via MQTT protocol. The satellites will forward this processed data to AWS IoT Core through the Iridium ground stations, which will act as a gateway for IoT based data. From this point, the threshold will be compared with the values stored in DynamoDB

and a Lambda function will trigger, generating an SMS via SNS topics to alert the authorities so they can take appropriate action. The following are the advantages of the SMARTDART architecture over the traditional approach:
1. Latency will be reduced due to the data being processed at the edge of the network.
2. The online state of the buoys can easily be tracked on AWS IoT core dashboard. With inter-buoy communication via smart sensors, it will be easy to know which buoys are operative and which ones are not, thereby greatly improving the fault tolerance of the SMARTDART network.
3. Cost will be greatly reduced since the AWS databases are already set up for processing tsunami data, which eliminates upfront expenses needed for building base stations or tsunami centers. Also, with AWS's pay-as-you-go model, the charges, which are minimal as well, will only incur for the number of actions the sensors perform and the number of times the data is queried.
4. The SMARTDART system will be highly scalable compared to the current DART network since more smart buoys can be deployed at lower costs. The serverless cloud architecture contributes to scalability by providing elastic resource management, which means incase of a sudden surge in large volumes of data, the Lambda function can auto-scale.
Figure 4 shows the architecture of the SMARTDART solution.

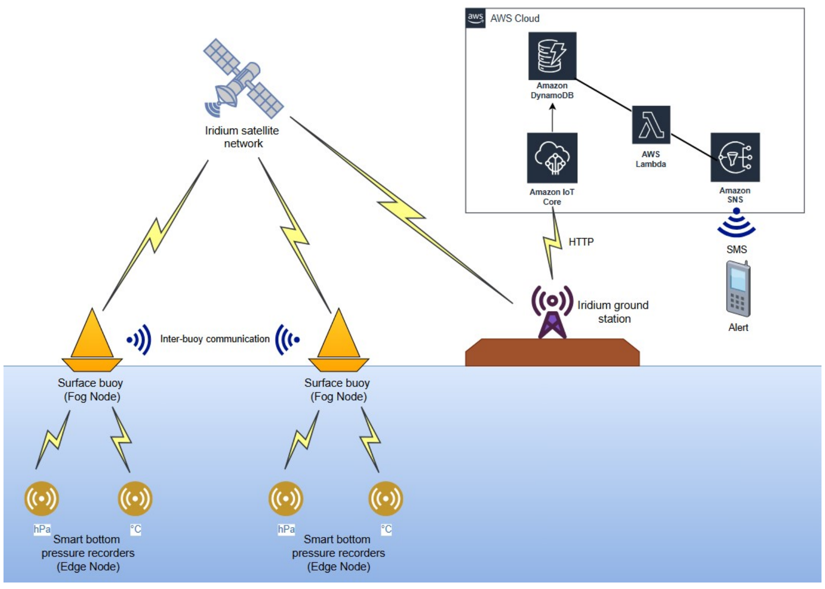
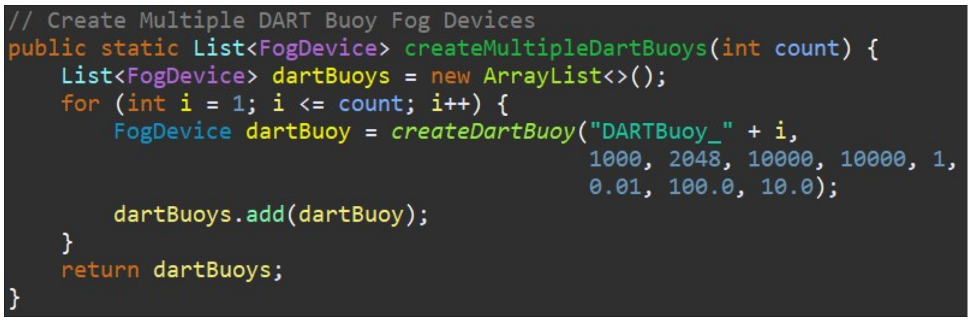


Figure 4: Practical application of SMARTDART

## 5 Implementation

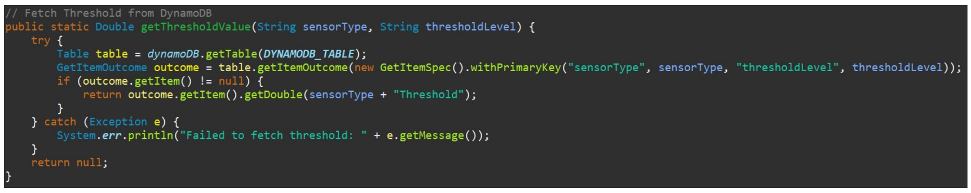
As mentioned in the previous sections, this project was developed using Java programming language in Eclipse IDE with iFogSim as the toolkit to simulate a fog and edge IoT environment of bottom pressure recorders. Two Java files have been created. The first file consists of the actual simulation code that mimics pressure and temperature sensors. The second file is a Lambda handler file that monitors the simulation's output and triggers an SNS topic when the threshold levels exceed. With real-world application in mind, the code allows the development of multiple fog devices (dart buoys) that must operate under varying network conditions. Figure 5 shows the code snippet for the configuration of multiple dart buoys.



```java
// Create Multiple DART Buoy Fog Devices
public static List<FogDevice> createMultipleDartBuoys(int count) {
    List<FogDevice> dartBuoys = new ArrayList<>();
    for (int i = 1; i <= count; i++) {
        FogDevice dartBuoy = createDartBuoy("DARTBuoy_" + i,
                                            1000, 2048, 10000, 10000, 1,
                                            0.01, 100.0, 10.0);
        dartBuoys.add(dartBuoy);
    }
    return dartBuoys;
}
```

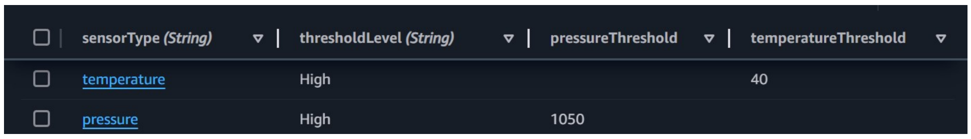
Figure 5: Creating multiple DART buoys

The sensor reading are generated randomly. Threshold values stored in DynamoDB are fetched for comparison with the help of the static method *getThresholdValue*. Figure 6 shows the code snippet of the DynamoDB method and figure 7 shows the DynamoDB table that stores the threshold values.



```java
// Fetch Threshold from DynamoDB
public static Double getThresholdValue(String sensorType, String thresholdLevel) {
    try {
        Table table = dynamoDB.getTable(DYNAMODB_TABLE);
        GetItemOutcome outcome = table.getItemOutcome(new GetItemSpec().withPrimaryKey("sensorType", sensorType, "thresholdLevel", thresholdLevel));
        if (outcome.getItem() != null) {
            return outcome.getItem().getDouble(sensorType + "Threshold");
        }
    } catch (Exception e) {
        System.err.println("Failed to fetch threshold: " + e.getMessage());
    }
    return null;
}
```

Figure 6: DynamoDB method to fetch threshold values

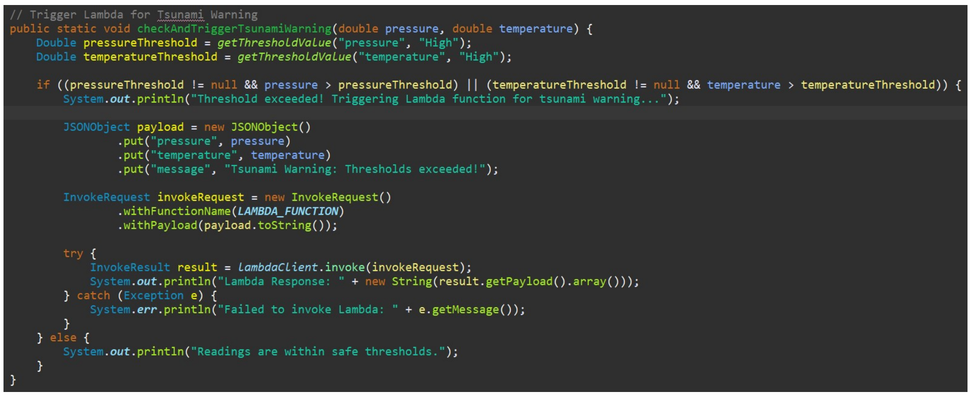


| sensorType (String) | thresholdLevel (String) | pressureThreshold | temperatureThreshold |
|---|---|---|---|
| temperature | High | | 40 |
| pressure | High | 1050 | |

Figure 7: DynamoDB table that stores the threshold values

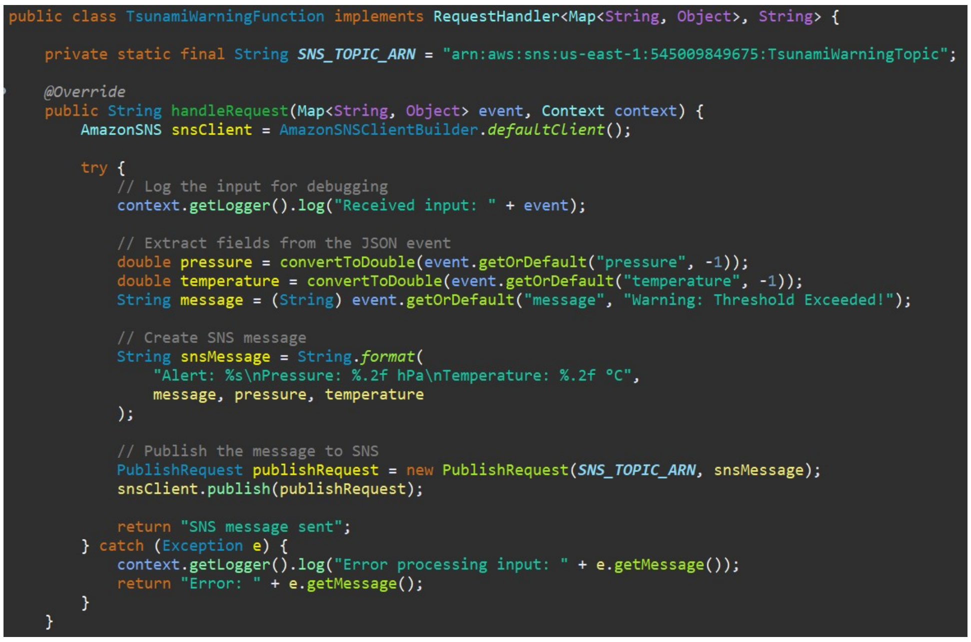
The Lambda function to trigger based on exceeding threshold levels is defined by the *checkAndTriggerTsunamiWarning* static method. It is invoked only when the output crosses both the pressure and the temperature thresholds. The public class *Tsunami-WarningFunction* is responsible for communicating with the SNS topic to send alerts. The message to be sent is configured within this class, which consists of an alert warning and the pressure/temperature values that generated the alert. Figure 8 shows the code snippet of the Lambda method and Figure 9 shows the *TsunamiWarningFunction* class code.

The simulation generates output in the form of readings from both the sensors along with the performance metrics, these being the latency for each reading of pressure and temperature data sent and when the simulation is completed, the output displays the



```java
// Trigger Lambda for Tsunami Warning
public static void checkAndTriggerTsunamiWarning(double pressure, double temperature) {
    Double pressureThreshold = getThresholdValue("pressure", "High");
    Double temperatureThreshold = getThresholdValue("temperature", "High");

    if ((pressureThreshold != null && pressure > pressureThreshold) || (temperatureThreshold != null && temperature > temperatureThreshold)) {
        System.out.println("Threshold exceeded! Triggering Lambda function for tsunami warning...");

        JSONObject payload = new JSONObject()
                .put("pressure", pressure)
                .put("temperature", temperature)
                .put("message", "Tsunami Warning: Thresholds exceeded!");

        InvokeRequest invokeRequest = new InvokeRequest()
                .withFunctionName(LAMBDA_FUNCTION)
                .withPayload(payload.toString());

        try {
            InvokeResult result = lambdaClient.invoke(invokeRequest);
            System.out.println("Lambda Response: " + new String(result.getPayload().array()));
        } catch (Exception e) {
            System.err.println("Failed to invoke Lambda: " + e.getMessage());
        }
    } else {
        System.out.println("Readings are within safe thresholds.");
    }
}
```

Figure 8: Lambda method triggers



```java
public class TsunamiWarningFunction implements RequestHandler<Map<String, Object>, String> {

    private static final String SNS_TOPIC_ARN = "arn:aws:sns:us-east-1:545009849675:TsunamiWarningTopic";

    @Override
    public String handleRequest(Map<String, Object> event, Context context) {
        AmazonSNS snsClient = AmazonSNSClientBuilder.defaultClient();

        try {
            // Log the input for debugging
            context.getLogger().log("Received input: " + event);

            // Extract fields from the JSON event
            double pressure = convertToDouble(event.getOrDefault("pressure", -1));
            double temperature = convertToDouble(event.getOrDefault("temperature", -1));
            String message = (String) event.getOrDefault("message", "Warning: Threshold Exceeded!");

            // Create SNS message
            String snsMessage = String.format(
                "Alert: %s\nPressure: %.2f hPa\nTemperature: %.2f °C",
                message, pressure, temperature
            );

            // Publish the message to SNS
            PublishRequest publishRequest = new PublishRequest(SNS_TOPIC_ARN, snsMessage);
            snsClient.publish(publishRequest);

            return "SNS message sent";
        } catch (Exception e) {
            context.getLogger().log("Error processing input: " + e.getMessage());
            return "Error: " + e.getMessage();
        }
    }
```

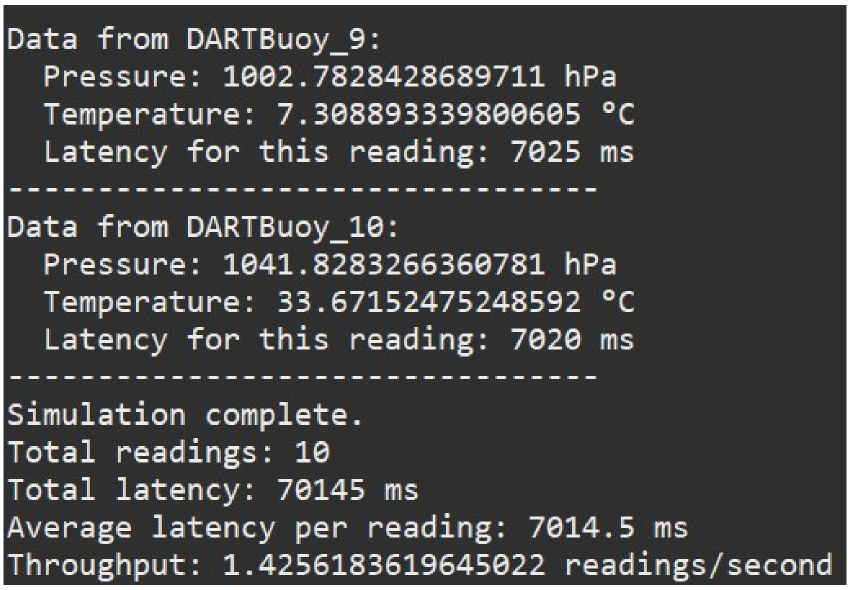
Figure 9: Lambda code for communication with SNS

total latency, average latency and throughput. When the threshold level exceeds, as shown in the Figure 10, a text is sent to a pre-defined endpoint via SNS. In this case, the endpoint is a mobile number. Figure 11 shows the text that will alert the authorities of a possible tsunami hazard.

The technical architecture of this project is based on simulations of real-world sensors. During actual implementation, other factors, such as hardware specifications, network types, communication protocols, cost of sustainability, etc will be needed to be taken in account.

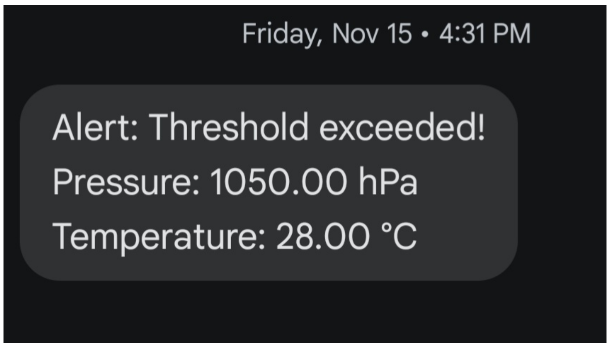
## 6 Evaluation

For the purpose of fair comparison, an additional simulation model was developed that functions like the real-world DART network. This model, called DART for the sake of



```
Data from DARTBuoy_9:
  Pressure: 1002.7828428689711 hPa
  Temperature: 7.308893339800605 °C
  Latency for this reading: 7025 ms
---------------------------------
Data from DARTBuoy_10:
  Pressure: 1041.8283266360781 hPa
  Temperature: 33.67152475248592 °C
  Latency for this reading: 7020 ms
---------------------------------
Simulation complete.
Total readings: 10
Total latency: 70145 ms
Average latency per reading: 7014.5 ms
Throughput: 1.4256183619645022 readings/second
```

Figure 10: Sensor readings and metrics



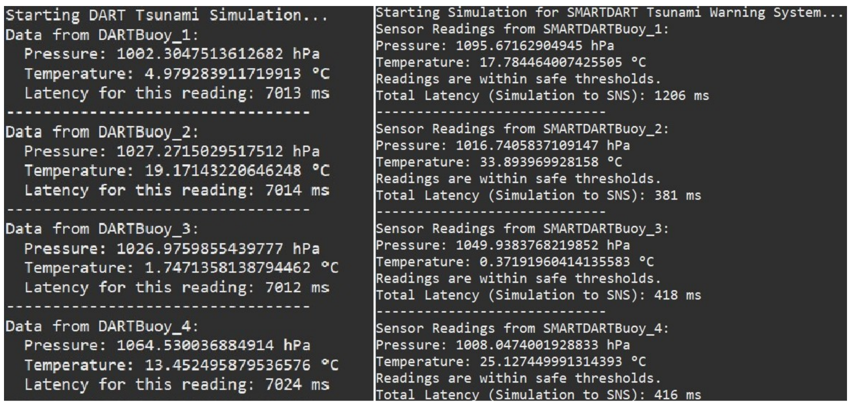
Friday, Nov 15 • 4:31 PM

Alert: Threshold exceeded!
Pressure: 1050.00 hPa
Temperature: 28.00 °C

Figure 11: Alert sent to a mobile phone

simplicity, is basic without any external enhancements unlike the proposed solution that is cloud-based. Both the models output 10 samples of data each in short bursts of 15 second intervals. In case of the DART model, an end-to-end latency of 7 seconds (7000 milliseconds) has been defined. This number was chosen after reviewing the experiments performed by McMahon and Rathburn (2005) to measure the time it takes for data packets to travel to and return back to the ground station from the Iridium satellite using Short Burst Data (SBD) transfer, which is how data is sent throughout the actual DART network. The SMARTDART model has not been configured with pre-defined latency since its latency is measured by the end-to-end data transfer from the moment the simulation runs to the moment the alert is received on the phone via SNS. Both the simulations run for 3 minutes with sensor data transfer occurring in 15-second intervals, taking 10 reading in total and displaying them each in the output. Figure. 12 shows the readings of DART and SMARTDART models side-by-side.

Three experiments have been performed simulating 10 buoys each. For both the models, the output displays their total latencies and their throughput as performance metrics and these are the metrics that will be analyzed for their performance as they reflect the



```
Starting DART Tsunami Simulation...
Data from DARTBuoy_1:
  Pressure: 1002.3047513612682 hPa
  Temperature: 4.979283911719913 °C
  Latency for this reading: 7013 ms
---------------------------------
Data from DARTBuoy_2:
  Pressure: 1027.2715029517512 hPa
  Temperature: 19.17143220646248 °C
  Latency for this reading: 7014 ms
---------------------------------
Data from DARTBuoy_3:
  Pressure: 1026.9759855439777 hPa
  Temperature: 1.7471358138794462 °C
  Latency for this reading: 7012 ms
---------------------------------
Data from DARTBuoy_4:
  Pressure: 1064.530036884914 hPa
  Temperature: 13.452495879536576 °C
  Latency for this reading: 7024 ms
```

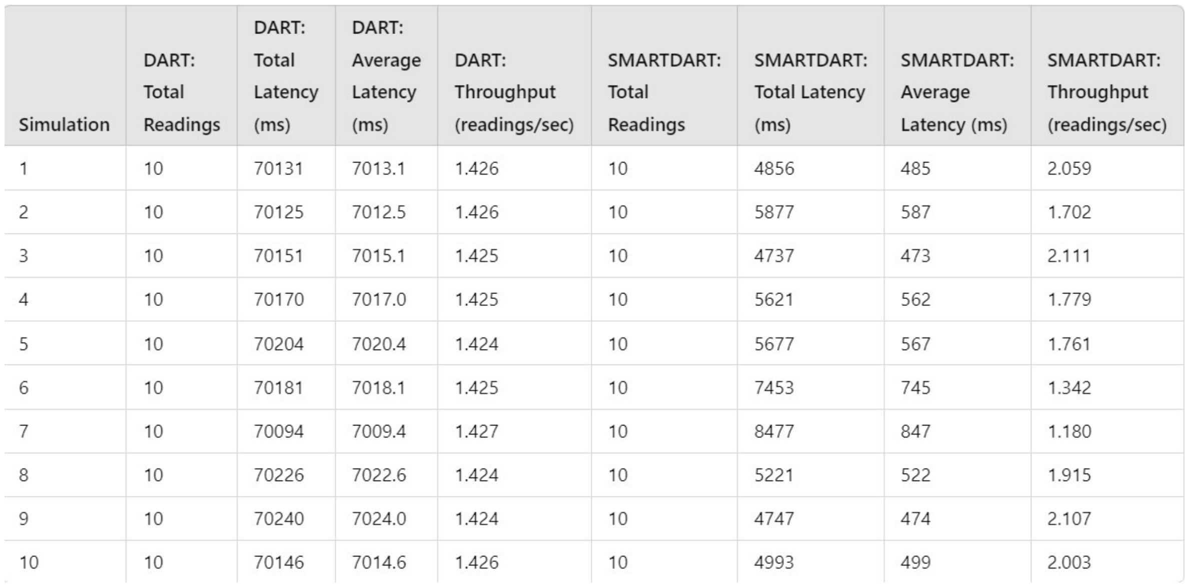
```
Starting Simulation for SMARTDART Tsunami Warning System...
Sensor Readings from SMARTDARTBuoy_1:
Pressure: 1095.67162904945 hPa
Temperature: 17.784464007425505 °C
Readings are within safe thresholds.
Total Latency (Simulation to SNS): 1206 ms
----------------------------
Sensor Readings from SMARTDARTBuoy_2:
Pressure: 1016.7405837109147 hPa
Temperature: 33.893969928158 °C
Readings are within safe thresholds.
Total Latency (Simulation to SNS): 381 ms
----------------------------
Sensor Readings from SMARTDARTBuoy_3:
Pressure: 1049.9383768219852 hPa
Temperature: 0.37191960414135583 °C
Readings are within safe thresholds.
Total Latency (Simulation to SNS): 418 ms
----------------------------
Sensor Readings from SMARTDARTBuoy_4:
Pressure: 1008.0474001928833 hPa
Temperature: 25.127449991314393 °C
Readings are within safe thresholds.
Total Latency (Simulation to SNS): 416 ms
```

Figure 12: DART and SMARTDART readings output

real-life functioning of the current system. The first experiment measures performance when no alerts are generated. The second experiment measures performance when alerts are being generated more often. For the third experiment, the DART and SMARTDART models were reprogrammed to simulate highly fluctuating/unstable network conditions similar to those found in extreme ocean environments.

## 6.1 Experiment 1

Figure. 13 shows the table containing metric data from 10 simulations of both models performed in a stable environment over a relatively stable network.



| Simulation | DART: Total Readings | DART: Total Latency (ms) | DART: Average Latency (ms) | DART: Throughput (readings/sec) | SMARTDART: Total Readings | SMARTDART: Total Latency (ms) | SMARTDART: Average Latency (ms) | SMARTDART: Throughput (readings/sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 70131 | 7013.1 | 1.426 | 10 | 4856 | 485 | 2.059 |
| 2 | 10 | 70125 | 7012.5 | 1.426 | 10 | 5877 | 587 | 1.702 |
| 3 | 10 | 70151 | 7015.1 | 1.425 | 10 | 4737 | 473 | 2.111 |
| 4 | 10 | 70170 | 7017.0 | 1.425 | 10 | 5621 | 562 | 1.779 |
| 5 | 10 | 70204 | 7020.4 | 1.424 | 10 | 5677 | 567 | 1.761 |
| 6 | 10 | 70181 | 7018.1 | 1.425 | 10 | 7453 | 745 | 1.342 |
| 7 | 10 | 70094 | 7009.4 | 1.427 | 10 | 8477 | 847 | 1.180 |
| 8 | 10 | 70226 | 7022.6 | 1.424 | 10 | 5221 | 522 | 1.915 |
| 9 | 10 | 70240 | 7024.0 | 1.424 | 10 | 4747 | 474 | 2.107 |
| 10 | 10 | 70146 | 7014.6 | 1.426 | 10 | 4993 | 499 | 2.003 |

Figure 13: Simulation of normal readings

The total latency has been calculated by the difference between the start time of the simulation and the action performed. In this experiment, since the Lambda function is not being invoked, the action is limited to simply querying the DynamoDB table. The average latency for each reading is calculated by dividing the total latency by the total number of readings:

$$AverageLatency = \frac{TotalLatency}{TotalReadings}$$

Throughput is the measure of the number of sensor readings being transmitted per second. In real-world DART network, higher throughput means more volume of data being processed, thereby increasing fault tolerance and accuracy. It is calculated by dividing the total readings by the total latency:

$$Throughput = \frac{TotalReadings}{TotalLatency}$$

Figures. 14 and 15 depict the data plotted on charts for total latencies and throughput, respectively, of both models. The charts were created in Python using matplotlib library for data visualization.

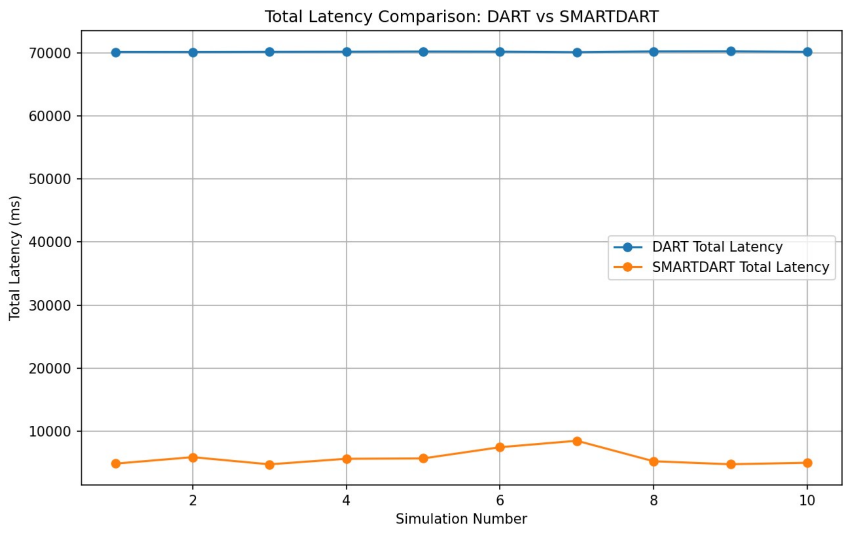


Figure 14: Total latencies of DART and SMARTDART under normal conditions

## 6.2 Experiment 2

The experiment was performed again, this time with a higher frequency of alert generation. This configuration simulates buoys present in tsunami-prone regions where the water pressure and temperature constantly undergoes abnormal levels of fluctuations but over a stable network. The end-to-end latencies (total and average) of the SMARTDART are measured from the time the simulation begins to the time when the text message is sent to a device through SNS. Figure. 16 shows the table with the simulation data of this experiment.

Figures. 17 and 18 shows the total latencies and throughput of both models respectively, plotted on charts.

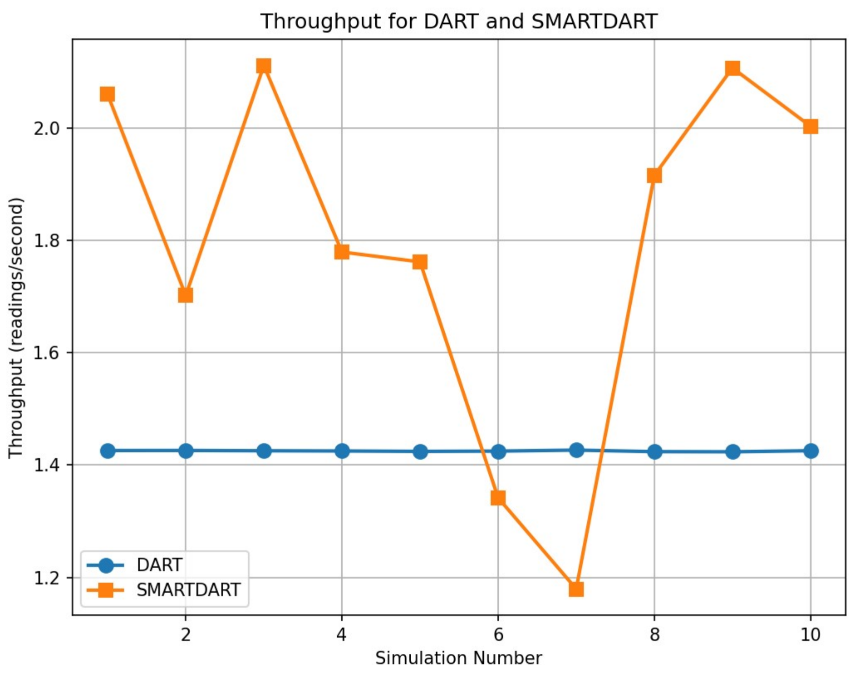
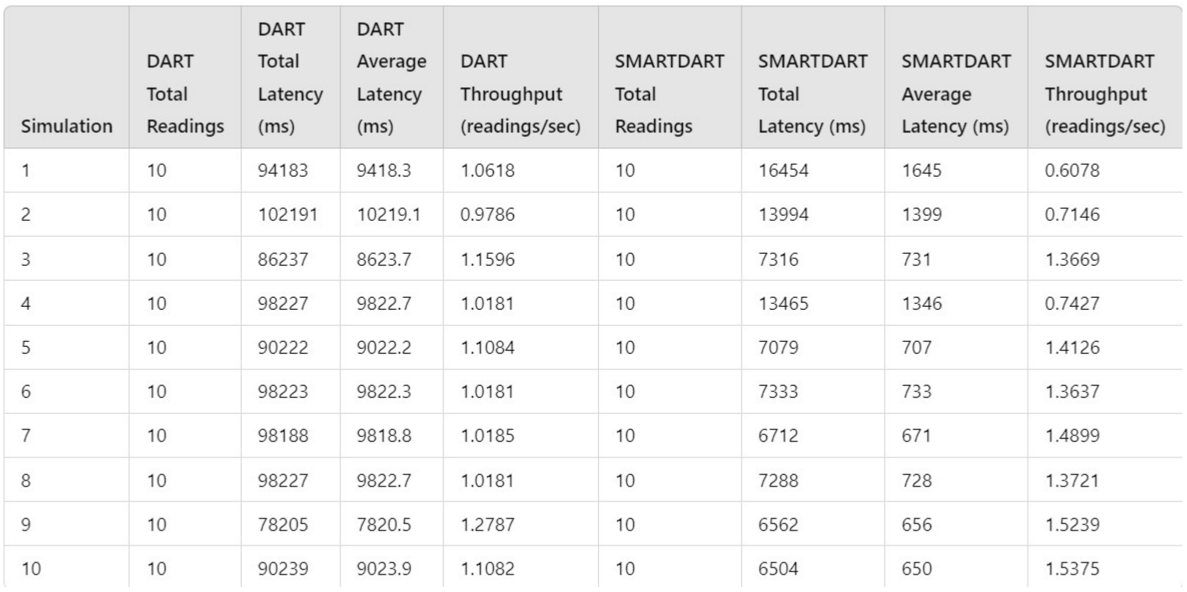


Figure 15: Throughput of DART and SMARTDART under normal conditions



| Simulation | DART Total Readings | DART Total Latency (ms) | DART Average Latency (ms) | DART Throughput (readings/sec) | SMARTDART Total Readings | SMARTDART Total Latency (ms) | SMARTDART Average Latency (ms) | SMARTDART Throughput (readings/sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 94183 | 9418.3 | 1.0618 | 10 | 16454 | 1645 | 0.6078 |
| 2 | 10 | 102191 | 10219.1 | 0.9786 | 10 | 13994 | 1399 | 0.7146 |
| 3 | 10 | 86237 | 8623.7 | 1.1596 | 10 | 7316 | 731 | 1.3669 |
| 4 | 10 | 98227 | 9822.7 | 1.0181 | 10 | 13465 | 1346 | 0.7427 |
| 5 | 10 | 90222 | 9022.2 | 1.1084 | 10 | 7079 | 707 | 1.4126 |
| 6 | 10 | 98223 | 9822.3 | 1.0181 | 10 | 7333 | 733 | 1.3637 |
| 7 | 10 | 98188 | 9818.8 | 1.0185 | 10 | 6712 | 671 | 1.4899 |
| 8 | 10 | 98227 | 9822.7 | 1.0181 | 10 | 7288 | 728 | 1.3721 |
| 9 | 10 | 78205 | 7820.5 | 1.2787 | 10 | 6562 | 656 | 1.5239 |
| 10 | 10 | 90239 | 9023.9 | 1.1082 | 10 | 6504 | 650 | 1.5375 |

Figure 16: Simulation of frequent readings

## 6.3 Experiment 3

The third experiment simulates the DART and SMARTDART models in high-alert environment with highly unstable network conditions. This is to test how these systems perform in the worst-case scenario. Figure. 19 shows the table with their metrics under these conditions.

Figure. 20 and 21 show the latencies and throughput of these two models plotted on line charts.

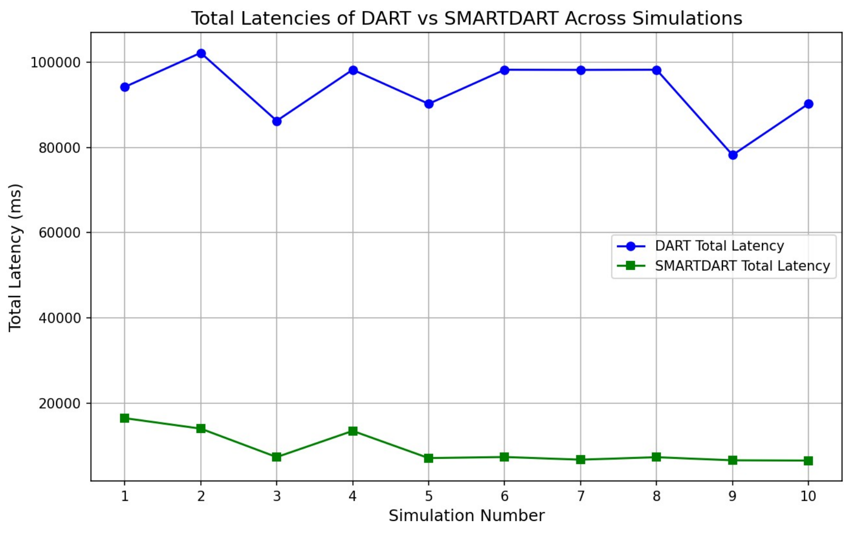


Figure 17: Total latencies of DART and SMARTDART under high alert conditions

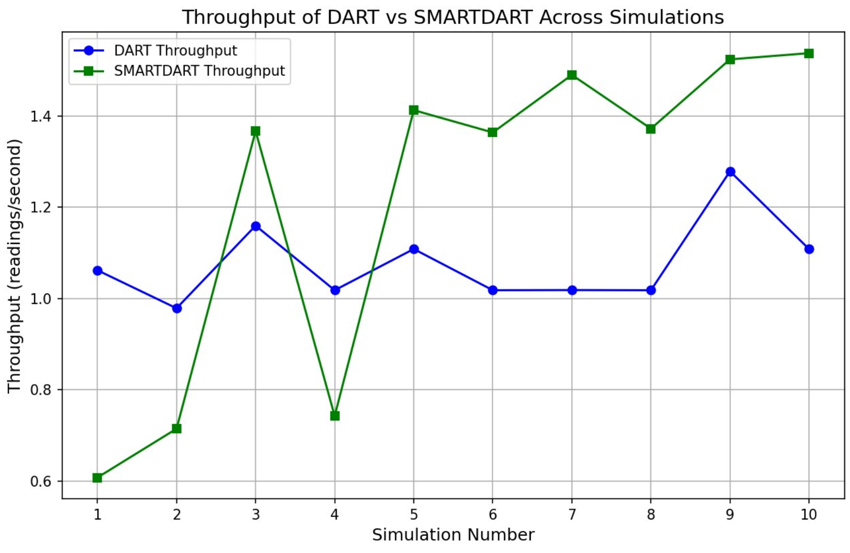


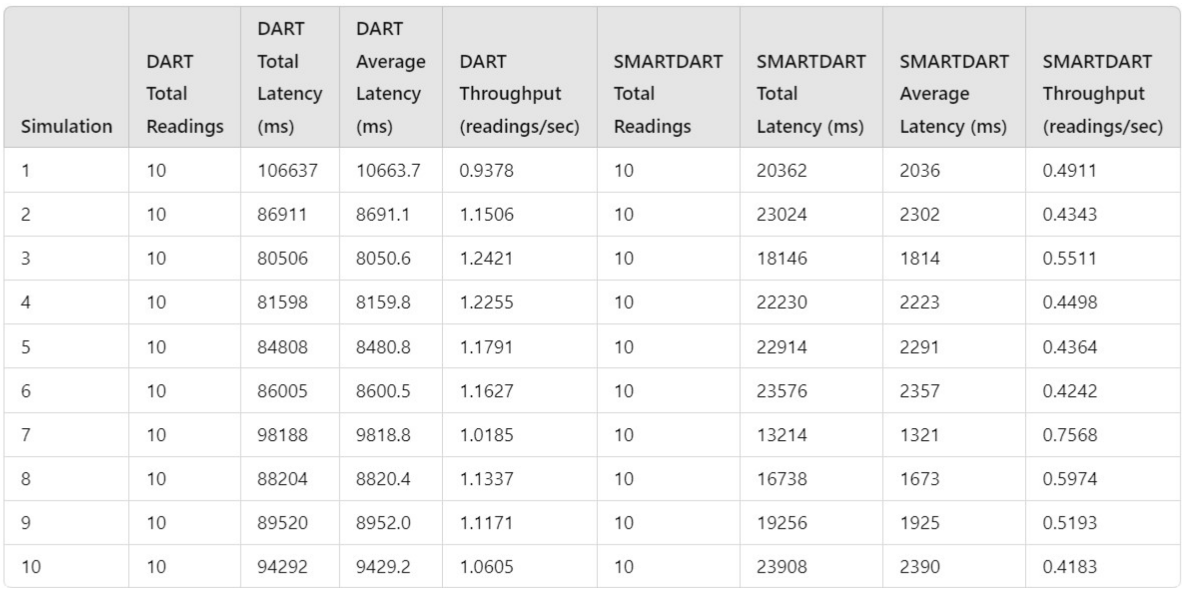
Figure 18: Throughput of DART and SMARTDART under high alert conditions

## 6.4 Discussion

The experiments yielded highly noticeable difference in the performances of DART and SMARTDART. Notably, the SMARTDART model outperformed the DART by a major margin.

### 6.4.1 Experiment 1

This experiment was performed under stable network conditions without triggering any alerts. The analysis of the table shows that the total latency of DART was significantly higher than that of SMARTDART's. The average total latency of DART was 70,166.8 milliseconds while that of the SMARTDART was 5,765.9 milliseconds. This is a difference of over 91.78%, which was also reflected in the average latencies per reading for each buoys of both models. The throughput values averaged at 1.425 readings/second for



| Simulation | DART Total Readings | DART Total Latency (ms) | DART Average Latency (ms) | DART Throughput (readings/sec) | SMARTDART Total Readings | SMARTDART Total Latency (ms) | SMARTDART Average Latency (ms) | SMARTDART Throughput (readings/sec) |
|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 106637 | 10663.7 | 0.9378 | 10 | 20362 | 2036 | 0.4911 |
| 2 | 10 | 86911 | 8691.1 | 1.1506 | 10 | 23024 | 2302 | 0.4343 |
| 3 | 10 | 80506 | 8050.6 | 1.2421 | 10 | 18146 | 1814 | 0.5511 |
| 4 | 10 | 81598 | 8159.8 | 1.2255 | 10 | 22230 | 2223 | 0.4498 |
| 5 | 10 | 84808 | 8480.8 | 1.1791 | 10 | 22914 | 2291 | 0.4364 |
| 6 | 10 | 86005 | 8600.5 | 1.1627 | 10 | 23576 | 2357 | 0.4242 |
| 7 | 10 | 98188 | 9818.8 | 1.0185 | 10 | 13214 | 1321 | 0.7568 |
| 8 | 10 | 88204 | 8820.4 | 1.1337 | 10 | 16738 | 1673 | 0.5974 |
| 9 | 10 | 89520 | 8952.0 | 1.1171 | 10 | 19256 | 1925 | 0.5193 |
| 10 | 10 | 94292 | 9429.2 | 1.0605 | 10 | 23908 | 2390 | 0.4183 |

Figure 19: Simulation of frequent readings under unstable network

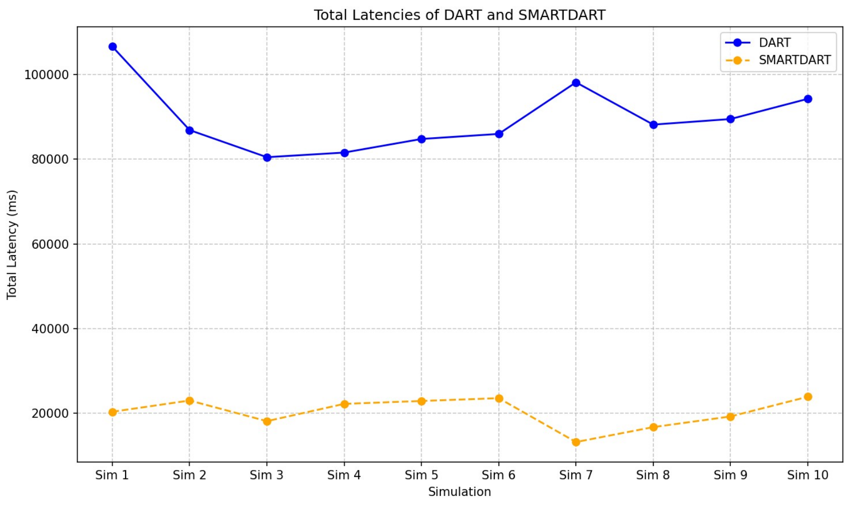


Figure 20: Total latencies of DART and SMARTDART under high alert and unstable network conditions

DART and 1.796 readings/second for SMARTDART, which means that SMARTDART generated approximately 26.01% more readings per second than DART.

### 6.4.2 Experiment 2

This experiment was performed under high-alert conditions with readings frequently crossing threshold levels. The analysis of the table showed that the total latency of DART was still higher than that of SMARTDART. In this experiment, the average total latency of DART was 93,314.2 milliseconds while SMARTDART's total latency averaged at 9,270.7 milliseconds, making it approximately 90% faster. The throughput of DART averaged at 1.077 readings/second while that of SMARTDART averaged at 1.213 readings/second, displaying an improvement of approximately 13% over DART. This showed

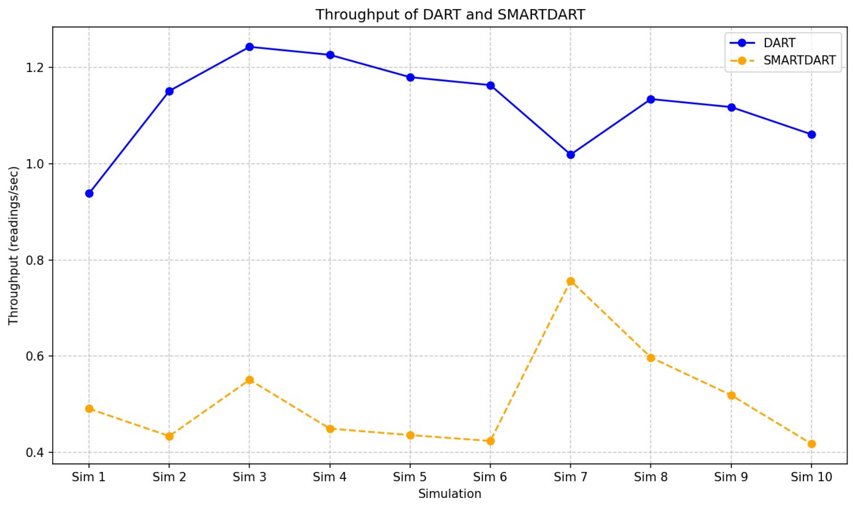


Figure 21: Throughput of DART and SMARTDART under high alert and unstable network conditions

that under high-alert environments, throughput drops while latency increases for both models.

### 6.4.3 Experiment 3

This experiment was performed to simulate the worst case scenario, which was a high-alert environment with unstable network conditions. In this experiment, the latencies rose considerable compared to the previous two experiments, which could be attributed to the environment and network. DART showed an average total latency of 87,916.9 milliseconds while SMARTDART's total latency averaged at 20,336.8 milliseconds. SMARTDART outperformed DART by 76.87%, which is a major drop from the previous performances that were 90% and over. Interestingly in this experiment, DART had a higher throughput than SMARTDART, the former generating 1.123 readings/second while the latter generating 0.509 readings/second, making it generate 54.67% more readings than SMARTDART.

## 6.5 Analysis

The results from the three experiments proved that SMARTDART offered much lower latency than DART with improvements ranging from 76% on the lower end to over 90% on the higher end. The same went to the average latencies per readings for both models. This was attributed to the integration with AWS and its services. Similarly, SMARTDART produced more readings per second than DART in the first two experiments although the improvements were minimal, ranging from 13% to 26%. It is worth noting, however, that these experiments were performed strictly in a simulation environment with pre-defined parameters based on the literature review for the DART model, which is one reason for the performance it showed. SMARTDART, when practically implemented, will have an additional MQTT network layer and AWS IoT Core service for receiving the processed sensor data, which could increase latency. But due to data being processed at the buoys themselves, which will act as fog devices and AWS's highly reliable infrastructure for IoT technology compared to other cloud vendors based on the research in the literature

review, the overall total latency of SMARTDART is still expected to be lower than that of the current DART system's 3-minute round trip time from the sensors to the tsunami warning centers after a trigger is generated.
The current DART buoys are equipped with Motorola 9522 L-band transceiver, Leadtek model 9546 receiver GPS and Motorola 68332 32-bit micro-controller. Each buoy costs $250,000 to purchase, according to Bernard et al. (2001). A majority of this price consists of expensive components in the buoys and the tsunameters. SMARTDART, however, will consist of smart sensors that are highly compatible with IoT applications at lower prices. NVIDIA Jetson NANO, a high-performance real-time data processing microcomputer which is also capable of supporting AI-based detection, costs only about $225 Doan and Phan (2024). Likewise, Iridium Edge, a satellite-IoT communications device, costs between $350 and $450 only with plans ranging from $15 per month for 8 KB data to $56 per month for 30 KB, based on the current price trends from e-commerce websites. While the bottom pressure recorders will remain the same for accurate data gathering, the other components could be upgraded with more affordable IoT compatible technologies compared to those found in the current DART buoys. AWS's pay-as-you-go model ensure charges are incurred only for the amount of time the functions of the buoys are in action, which makes it even more affordable compared to the high infrastructure costs of setting up tsunami centers.

## 7 Conclusion and Future Work

The objective of this research was to explore how integrating cloud technology into DART tsunami warning system will enhance its capabilities. An AWS-based simulation model, named SMARTDART, was developed in iFogSim and its performance was compared with a base DART model in three experiments simulating three different scenarios. In almost all the experiments, the SMARTDART model showed much lower latencies and higher throughput than the DART model. The output of the experiments proved that SMARTDART is capable of performing exceptionally well in both low stress and high stress environments. A practical SMARTDART model was also presented that described how the system will work in the real world. One limitation of this project was the absence of an additional MQTT network layer which could have taken its functionality even closer to a potential real-world project. Nonetheless, the research proved that its application is possible and that cloud integration will improve the current DART system while making it a cost-effective solution in the long term.
In future, more effort will be put into building a live SMARTDART model with real sensors and communication protocols, along with a live mobile/desktop application that will allow authorities to keep track of tsunami events in real-time from the data being processed by the buoys. Moreover, this research has opened another gap for the possibility of implementing machine learning and AI into tsunami warning systems to improve the accuracy of their sensor readings and analyze patterns which could help predict a tsunami before it occurs, giving the authorities a better chance of taking appropriate actions to save lives.

## References

Abdelaziz, A., Mesbah, S. and Kholief, M. (2024). Innovative cost-efficient cloud computing-based models for disasters management, *Journal of Advanced Research in Applied Sciences and Engineering Technology* **48**(1): 100–116.

Adiprabowo, T., Ramdani, D., Daud, P., Andriana, A., Ali, E., Nasrullah, N. and Zulkarnain, Z. (2024). Radar-based tsunami detection: A comprehensive review, *Internet of Things and Artificial Intelligence Journal* **4**(2): 299–315.

Angove, M. D., Rabenold, C. L., Weinstein, S. A., Eblé, M. C. and Whitmore, P. M. (2015). U.s. tsunami warning system: Capabilities, gaps, and future vision, *OCEANS 2015 - MTS/IEEE Washington*, pp. 1–5.

Awaisi, K. S., Abbas, A., Khan, S. U., Mahmud, R. and Buyya, R. (2021). Simulating fog computing applications using ifogsim toolkit, *Mobile edge computing* pp. 565–590.

Bernard, E. N., González, F. I., Meinig, C. and Milburn, H. B. (2001). Early detection and real-time reporting of deep-ocean tsunamis, *International Tsunami Symposium*, pp. 7–10.

Doan, T.-N. and Phan, T.-H. (2024). A novel smart system with jetson nano for remote insect monitoring, *International Journal of Advanced Computer Science & Applications* **15**(7).

Esposito, M., Palma, L., Belli, A., Sabbatini, L. and Pierleoni, P. (2022). Recent advances in internet of things solutions for early warning systems: A review, *Sensors* **22**(6): 2124.

Gonzalez, F. I., Milburn, H. M., Bernard, E. N. and Newman, J. C. (1998). Deep-ocean assessment and reporting of tsunamis (dart): Brief overview and status report, *Proceedings of the international workshop on Tsunami Disaster Mitigation*, Vol. 19, NOAA Tokyo, Japan, p. 2.

Krichen, M., Abdalzaher, M. S., Elwekeil, M. and Fouda, M. M. (2024). Managing natural disasters: An analysis of technological advancements, opportunities, and challenges, *Internet of Things and Cyber-Physical Systems* **4**: 99–109.

Kusuma, A. A., Agastani, T., Nugroho, T., Anggraeni, S. P. and Hartawan, A. R. (2022). Estimating mqtt performance in a virtual testbed of ina-cbt communication sub-system, *2022 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, IEEE, pp. 73–77.

Lawson, R. A., Graham, D., Stalin, S., Meinig, C., Tagawa, D., Lawrence-Slavas, N., Hibbins, R. and Ingham, B. (2012). Next generation easy-to-deploy (etd) tsunami assessment buoy, *2012 Oceans - Yeosu*, pp. 1–9.

McClure, M., Sabine, C., Feely, R., Hammond, S., Meinig, C., McPhaden, M., Stabeno, P. and Bernard, E. (2023). 50 years of pmel tsunami research and development.

McMahon, M. M. and Rathburn, R. (2005). Measuring latency in iridium satellite constellation data services, *US Naval Academy Report* (A291464): 68.

Muhammed, A. S. and Ucuz, D. (2020). Comparison of the iot platform vendors, microsoft azure, amazon web services, and google cloud, from users' perspectives, *2020 8th International Symposium on Digital Forensics and Security (ISDFS)*, pp. 1–4.

Naas, M. I., Boukhobza, J., Raipin Parvedy, P. and Lemarchand, L. (2018). An extension to ifogsim to enable the design of data placement strategies.

Percival, D. B., Denbo, D. W., Gica, E., Huang, P. Y., Mofjeld, H. O., Spillane, M. C. and Titov, V. V. (2016). Evaluating effectiveness of dart buoy networks, *arXiv preprint arXiv:1607.02795* .

Qiu, M., Ming, Z., Wang, J., Yang, L. T. and Xiang, Y. (2014). Enabling cloud computing in emergency management systems, *IEEE Cloud Computing* **1**(4): 60–67.

Siek, M. and Larry, L. (2021). Design and implementation of internet of things and cloud technology in flood risk mitigation, *2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS)*, pp. 1–6.

Srivihok, P., Honda, K., Ruangrassamee, A., Muangsin, V., Naparat, P., Foytong, P., Promdumrong, N., Aphimaeteethomrong, P., Intavee, A., Layug, J. et al. (2014). Development of an online tool for tsunami inundation simulation and tsunami loss estimation, *Continental Shelf Research* **79**: 3–15.

Taft, B., Burdette, M., Riley, R., Hansen, B., Wells, W., Maxwell, D. and Mettlach, T. (2009). Development of an ndbc standard buoy, *OCEANS 2009*, IEEE, pp. 1–10.

Ujjwal, K., Garg, S., Hilton, J., Aryal, J. and Forbes-Smith, N. (2019). Cloud computing in natural hazard modeling systems: Current research trends and future directions, *International Journal of Disaster Risk Reduction* **38**: 101188.

Wang, Y., Jiménez, C., Quiroz, M. and Ortega, E. (2023). Tsunami early warning system using offshore tsunameters in peru, *Ocean Engineering* **279**: 114516.

Yousuf Khan, E. U. and Rahim Soomro, T. (2022). Overview of ifogsim: A tool for simulating fog networks of the future, *2022 14th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS)*, pp. 1–4.

Živić, M., Nemec, D. and Bojović, Ž. (2023). Mqtt protocol in iot environment: Comparison with coap and zeromq protocols, *2023 31st Telecommunications Forum (TELFOR)*, IEEE, pp. 1–4.