**ISRO – Telemetry Tracking and Command Network (ISTRAC)**



**TITLE: COMPARISON OF OPTIMISATION ALGORITHMS IN CNNs IN MALWARE CLASSIFICATION ON THE MALEVIS DATASET**

UNDER THE GUIDANCE OF

Bharat Devasani

Scientist/Engineer-SE, NSA, ISTRAC

**SUBMITTED BY**

**Hrishikesh H Chandra Reg No: 01JST21IS017**

**STUDENT OF**

**JSS SCIENCE AND TECHNOLOGY UNIVERSITY – JSS STU**

**JSS TECHNICAL INSTITUTIONS CAMPUS, MYSORE - 57006**

**ACKNOWLEDGEMENT**

a

If words are considered as symbols of approval and tokens of acknowledgement, then let the words play the heralding role of expressing my gratitude to all those who have directly or indirectly helped during my period of internship.

a

I am delighted to thank **Dr. A. K. ANILKUMAR, Director, ISRO Telemetry, Tracking, and Command Network (ISTRAC), Bengaluru, Karnataka,** for giving me permission to pursue my Internship in ISTRAC, BENGALURU.

a

I express my gratitude to **Mr. SANKAR MADASWAMY B, Scientist/Engineer-SG, Manager, HRDD, ISTRAC,** for permitting me to undertake my internship.

a

My sincere thanks also goes to my guide **Mr. Bharath Devasani, Scientist/Engineer-SE, ISTRAC,** for his mentorship, constructive feedback, and technical insights. His meticulous approach and encouragement have greatly enhanced my understanding and technical proficiency.

a

I sincerely thank you all whole-heartedly for sparing your valuable time quite often, despite the heavy workload. The discussions had provided me with valuable insights into the workings of ISRO. Finally, I would like to thank my people (my parents, friends and university) for their continuous support in every field of my life.

a

a

**HRISHIKESH H CHANDRA**

**USN: 01JST21IS017**



**BONAFIDE CERTIFICATE**

This is to certify that the Internship report entitled **“COMPARISON OF OPTIMISATION ALGORITHMS IN CNNs IN MALWARE CLASSIFICATION ON THE MALEVIS DATASET REPORT”** submitted by **HRISHIKESH H CHANDRA**, bearing **REG NO: 01JST21IS017** of **JSS SCIENCE AND TECHNOLOGY UNIVERSITY, MYSORE, KARNATAKA**, for partial fulfilment for the award of Degree of Bachelor of Engineering, in **INFORMATION SCIENCE AND ENGINEERING (ISE)** is a bonafide record of the internship carried out by him under my supervision in the year 2025.

**BHARAT DEVASANI**

**Scientist/Engineer-SE,**

**Table of Contents**

[1. Abstract 7](#_Toc196390705)

[2. About ISTRAC 8](#_Toc196390706)

[3. About the Dataset 9](#_Toc196390707)

[3.1 Dataset Format and Pre-processing Steps 9](#_Toc196390708)

[3.2 Justification for Dataset Selection 10](#_Toc196390709)

[4. Introduction 11](#_Toc196390710)

[4.1 Background on Malware Detection and Static Analysis Using Computer Vision 11](#_Toc196390711)

[4.2 Overview of Convolutional Neural Networks (CNNs) in Malware Classification 12](#_Toc196390712)

[4.3 Project Objective 12](#_Toc196390713)

[4.4 Scope of the Project 13](#_Toc196390714)

[5. System Architecture 13](#_Toc196390715)

[5.1 High-Level Overview 13](#_Toc196390716)

[5.2 Data Flow Explanation 14](#_Toc196390717)

[6. NavIC Data Handling & Binary Log Decoding 16](#_Toc196390718)

[6.1 NavIC Log Structure 16](#_Toc196390719)

[6.2 Decoding Process for Range Measurement Data () 16](#_Toc196390720)

[6.3 Decoding Process for Navigation Data () 17](#_Toc196390721)

[6.4 Handling Data Fragmentation 17](#_Toc196390722)

[7. Implementation Details 19](#_Toc196390723)

[7.1 Multicast Sender – daps\_mcast\_sender.py 19](#_Toc196390724)

[7.2 Multicast Receiver – multicast\_rx.py 19](#_Toc196390725)

[7.3 Backend Server – app.py 20](#_Toc196390726)

[7.4 Frontend – index.html and script.js 21](#_Toc196390727)

[8. Testing and Results 22](#_Toc196390728)

[8.1 Testing Methodology 22](#_Toc196390729)

[8.2 Results Analysis 22](#_Toc196390730)

[8.3 Debugging and Iteration 26](#_Toc196390731)

[9. Challenges and Solutions 27](#_Toc196390732)

[9.1 Multi-Channel Processing 27](#_Toc196390733)

[9.2 Multicast Reception Issues 27](#_Toc196390734)

[9.3 Data Fragmentation and Reassembly 27](#_Toc196390735)

[10. Future Scope 28](#_Toc196390736)

[10.1 Expanding Station Support 28](#_Toc196390737)

[10.2 Database Integration 28](#_Toc196390738)

[10.3 Advanced Visualization 28](#_Toc196390739)

[10.4 AI-Based Anomaly Detection 28](#_Toc196390740)

[11. Conclusion 29](#_Toc196390741)

[11.1 Summary of Achievements 29](#_Toc196390742)

[11.2 Impact and Future Enhancements 29](#_Toc196390743)

[12. References 29](#_Toc196390744)

**Table of Figures**

[Figure 1 NavIC Architecture](about:blank) 9

[Figure 2 Multicast Data Flow](about:blank) 13

[Figure 3 Data Flow Diagram](about:blank) 14

[Figure 4 NavIC Log Structure](about:blank) 16

[Figure 5 Multicast Sender-Receiver Workflow](about:blank) 18

[Figure 6 Backend Data Processing](about:blank) 19

[Figure 7 Station 1 Range measurement Data](about:blank) 21

[Figure 8 Station 1 Navigation Data](about:blank) 22

[Figure 9 Station 2 Range measurement Data](about:blank) 22

[Figure 10 Station 2 Navigation Data](about:blank) 23

# Abstract

This report presents the design, implementation, and evaluation of a deep learning-based malware classification framework, developed using the MaleVis dataset, which comprises RGB visual representations of malware binaries. The project investigates the efficacy of convolutional neural networks (CNNs) in malware detection and conducts a comparative analysis of three optimization algorithms—Stochastic Gradient Descent (SGD), Adam, and Lion—on model performance and convergence behaviour.

The system architecture is structured around a training pipeline that processes the MaleVis dataset, which includes 25 malware families, and converts them into a structured format suitable for input into modern CNN models. DenseNet-121 was selected as the base architecture due to its proven performance in the referenced literature. The training and validation routines were implemented using PyTorch, and were configured to support training from scratch, avoiding pre-trained weights to match the original study's methodology.

A key contribution of this project is the extended experimentation with optimizers beyond the baseline SGD. Both Adam and Lion optimizers were integrated into the training pipeline and evaluated under identical training conditions. Performance metrics including accuracy, precision, and recall were used to assess the generalization ability of each optimizer. Comparative results showed notable variations in training speed, convergence rates, and final classification accuracy across the three optimizers.

Extensive testing was conducted to validate the system’s reliability and performance consistency on CPU-based execution environments. Additionally, the project addresses resource constraints such as GPU memory limitations and proposes solutions for training efficiency on lower-spec systems. Screenshots of training logs and metric graphs are included to demonstrate the experimental outcomes.

Future directions include extending the dataset with real-world samples, exploring data augmentation techniques, and deploying the trained model in a real-time malware detection system. Overall, this project offers a reproducible and extensible framework for static malware classification using vision-based deep learning techniques, providing insights into the impact of optimizer choice on model performance.

# About ISTRAC

ISTRAC (ISRO Telemetry Tracking and Command Network) is a ground segment network that provides support for the Indian Space Research Organisation's (ISRO) space missions:

* Telemetry, tracking, and command (TTC): ISTRAC provides TTC services from launch to satellite orbit injection. This includes tracking the launch vehicle from lift-off to spacecraft separation.
* Mission operations: ISTRAC carries out mission operations for remote sensing and scientific satellites.
* Spacecraft control centers: ISTRAC has established spacecraft control centers.
* Deep space network: ISTRAC has a deep space network that provides support for deep space missions.
* Weather radar design and development: ISTRAC designs and develops weather radars for launch vehicle tracking and meteorological applications.
* Search and rescue: ISTRAC provides search and rescue services.
* Space-based services: ISTRAC supports space-based services like telemedicine, tele-education, and Village Resource Centre (VRC).
* Space operations support: ISTRAC provides space operations support for other space agencies.

ISTRAC's headquarters are in Bangalore, Karnataka. It also has ground stations in Lucknow, Sriharikota, Thiruvananthapuram, Port Blair, Brunei, Biak (Indonesia), and Mauritius.

## About the Dataset

Satellites The MaleVis (Malware Evaluation with Vision) dataset is a specialized benchmark dataset designed to support research in vision-based malware classification. It consists of malware binaries that have been pre-processed and converted into RGB image representations, enabling the use of computer vision and deep learning techniques for static malware analysis. The dataset was introduced to facilitate experimentation with Convolutional Neural Networks (CNNs) in the domain of cyber security, particularly for identifying and classifying malicious software.

MaleVis contains images derived from 25 distinct malware families, with each sample corresponding to a Portable Executable (PE) file. The binary content of each PE file is interpreted as raw byte streams and visualized as images, preserving the structural patterns and byte-level sequences of the malware. Unlike traditional grayscale conversion methods used in earlier studies, MaleVis emphasizes 3-channel RGB image generation, which retains more detailed byte distribution characteristics and enables the use of more complex CNN architectures trained on colour image data.

The dataset is split into training and validation subsets, typically in a 70%-30% ratio, allowing for robust model evaluation. Each image in the dataset is labelled according to its malware family, and the balanced class distribution ensures fair training across categories.

This dataset plays a crucial role in bridging the gap between static malware detection and deep learning, offering researchers a realistic and scalable platform to evaluate classification performance, generalization, and the impact of various optimization techniques. Its design and structure are particularly suited for experimentation with modern deep learning models such as DenseNet, ResNet, and Inception, making it an ideal choice for this project’s comparative optimizer analysis.

## Dataset Format and Pre-processing Steps

The MaleVis dataset consists of RGB images generated from raw binary files of 25 different malware families. Each image is saved in a class-labelled directory structure, making it compatible with PyTorch’s ImageFolder loader.

The dataset is divided into two parts:

* Training set: 70% of the data, used for training the CNN model.
* Validation set: 30% of the data, used to evaluate the model’s generalization ability.

All images are resized to a standard resolution of 224×224 pixels to ensure compatibility with CNN architectures like ResNet and DenseNet.

The pre-processing pipeline applied before training includes the following transformations:

* Resize: All images are resized to 224×224 pixels.
* ToTensor: Images are converted into PyTorch tensors.
* Normalize: Images are normalized using mean = [0.5, 0.5, 0.5] and std = [0.5, 0.5, 0.5], as implemented in the actual training scripts. This normalization brings pixel values into a range of [−1, 1] and supports stable gradient updates during training.

These pre-processing steps are critical to ensure consistency across all optimizer and model variations used in the experiments. They were uniformly applied in each trial using SGD, Adam, and Lion optimizers.

## Justification for Dataset Selection

The MaleVis dataset was selected for this project due to its clear alignment with the goals of vision-based static malware classification using deep learning. It provides a robust benchmark for evaluating the effectiveness of convolutional neural networks (CNNs) by transforming binary malware files into structured RGB images. This format eliminates the need for handcrafted feature engineering and enables direct application of image classification techniques.

One of the key advantages of MaleVis is its balanced class distribution across 25 distinct malware families, allowing for fair and unbiased training. Each family is well-represented, which supports generalizable model learning and meaningful evaluation of classification accuracy. Additionally, the dataset is already partitioned into training and validation sets, following a standard 70-30 split, which simplifies model development and performance assessment.

Unlike some earlier malware datasets that rely on grayscale conversion or metadata analysis, MaleVis emphasizes pixel-level fidelity through its RGB representation, making it ideal for CNN-based exploration. Its compatibility with PyTorch’s ImageFolder class further facilitates easy integration with popular training pipelines.

Lastly, the MaleVis dataset has been referenced in peer-reviewed research, including the original paper replicated in this project. This validates its credibility and relevance as a research-grade dataset for experimentation and benchmarking in the malware detection domain.

## Introduction

## Background on Malware Detection and Static Analysis Using Computer Vision

Malware detection has traditionally relied on signature-based techniques, where known malicious code patterns are manually crafted and matched against incoming files. While effective for known threats, these methods fail to generalize to new, obfuscated, or polymorphic malware variants. As attackers increasingly use code mutation, packing, and encryption to evade detection, the need for more generalized and intelligent detection methods has become evident.

Static analysis, which inspects executable files without running them, offers a safer and often faster alternative to dynamic behavior analysis. One novel static approach treats malware binaries as structured data that can be transformed into visual representations—specifically, grayscale or RGB images. These visualizations are created by mapping the binary byte values directly to pixel intensities. This representation preserves structural features of the binary such as headers, code sections, and data patterns, which often exhibit consistent layouts across malware families.

The integration of computer vision with malware detection leverages the capability of Convolutional Neural Networks (CNNs) to automatically extract hierarchical features from images. By feeding malware-representative images into CNNs, the models can learn to differentiate between malware families based on texture, spatial arrangement, and frequency of byte patterns—similar to how they recognize objects in natural images.

This vision-based approach has shown promising results in classifying malware samples with high accuracy, especially when paired with robust datasets and well-optimized CNN architectures. In this project, this technique is expanded further to compare the performance of various optimization algorithms applied to CNN-based malware classifiers trained on the MaleVis dataset.

## Overview of Convolutional Neural Networks (CNNs) in Malware Classification

The Convolutional Neural Networks (CNNs) are a class of deep learning models that have revolutionized computer vision tasks such as image classification, object detection, and segmentation. CNNs are designed to automatically extract spatial and hierarchical features from images using a sequence of learnable filters applied across local receptive fields. These filters capture low-level patterns such as edges and textures in early layers, and more abstract, class-specific patterns in deeper layers.

In the context of malware classification, CNNs are employed to identify patterns in image representations of malware binaries. Instead of relying on handcrafted features or domain-specific heuristics, CNNs learn discriminative patterns directly from the raw pixel distributions of binary-to-image transformations. This allows the model to generalize across families of malware—even when variants are obfuscated or modified.

The core advantage of using CNNs for malware detection is their ability to learn from structured data representations without manual intervention. Given the complexity and variability of malware samples, especially with techniques like packing, polymorphism, and metamorphism, CNNs offer a scalable and automated alternative to traditional static analysis tools.

In this project, CNNs such as DenseNet121 and ResNet18 were used as core architectures. These models were trained from scratch on the MaleVis dataset, which contains RGB images of malware binaries. Each model learns to classify these images into one of 25 malware families based on spatial and visual characteristics embedded in the binary content.

The use of CNNs not only provides a powerful framework for malware detection but also opens up the possibility for further advancements such as explainable AI, transfer learning, and adversarial robustness in cybersecurity domains.

## Project Objective

* in a dynamic and user-friendly web interface.
* Support multiple stations (e.g., Station 1 and Station 2) within a single backend instance.

## Scope of the Project

This report details the design and implementation of the NavIC Ranging Data Monitoring System, which includes:

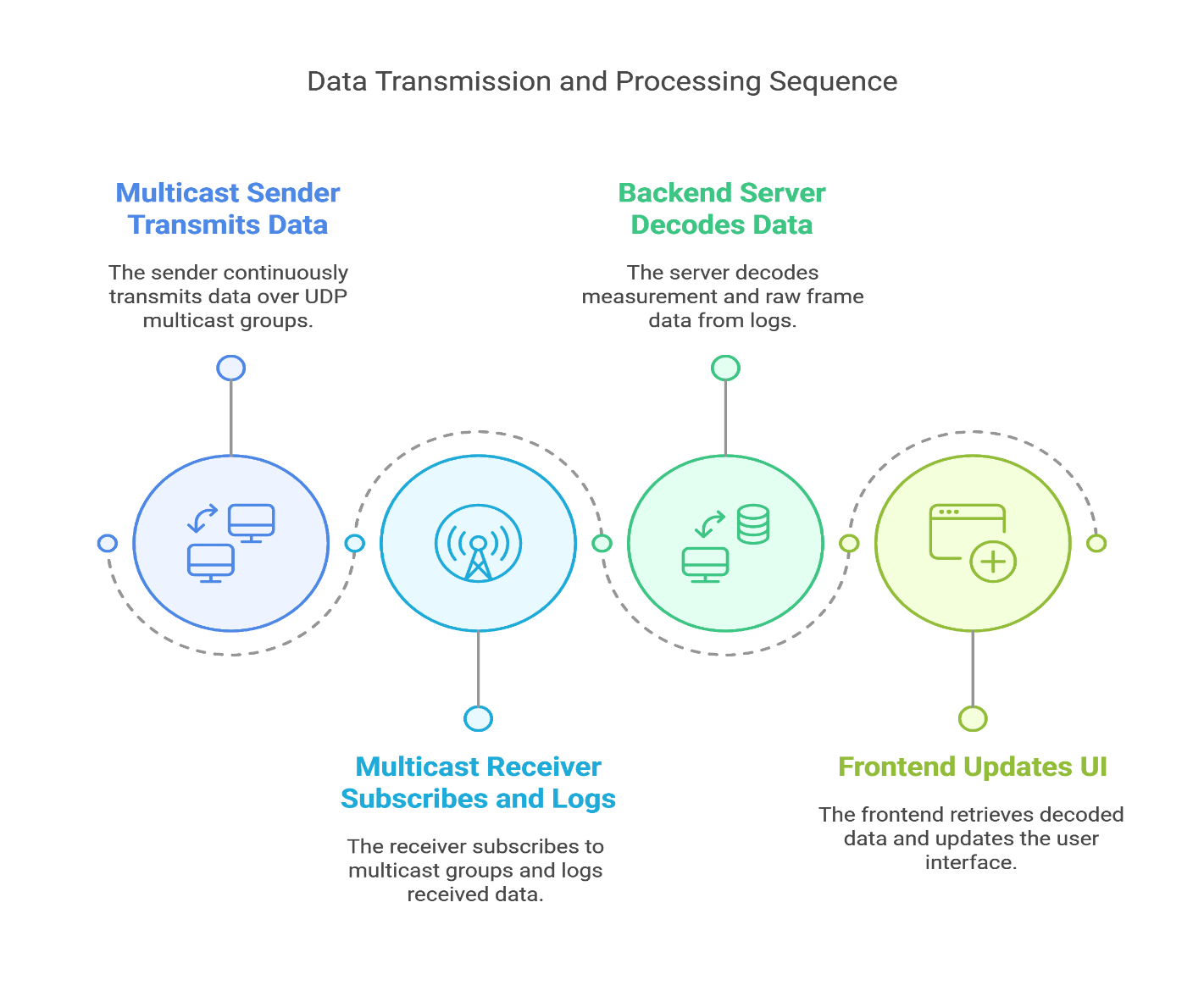
* **Data Acquisition:** Using multicast-based data transmission from the reference receivers.
* **Data Logging:** Capturing and storing binary logs at the ground stations.
* **Data Processing:** Decoding and parsing binary log files to extract range measurement and navigation information.
* **Visualization:** Presenting real-time data via a web-based user interface.
* **Multi-Channel Support:** Simultaneous processing of data from different NavIC channels (e.g., Station 1 and Station 2).

# System Architecture

## High-Level Overview

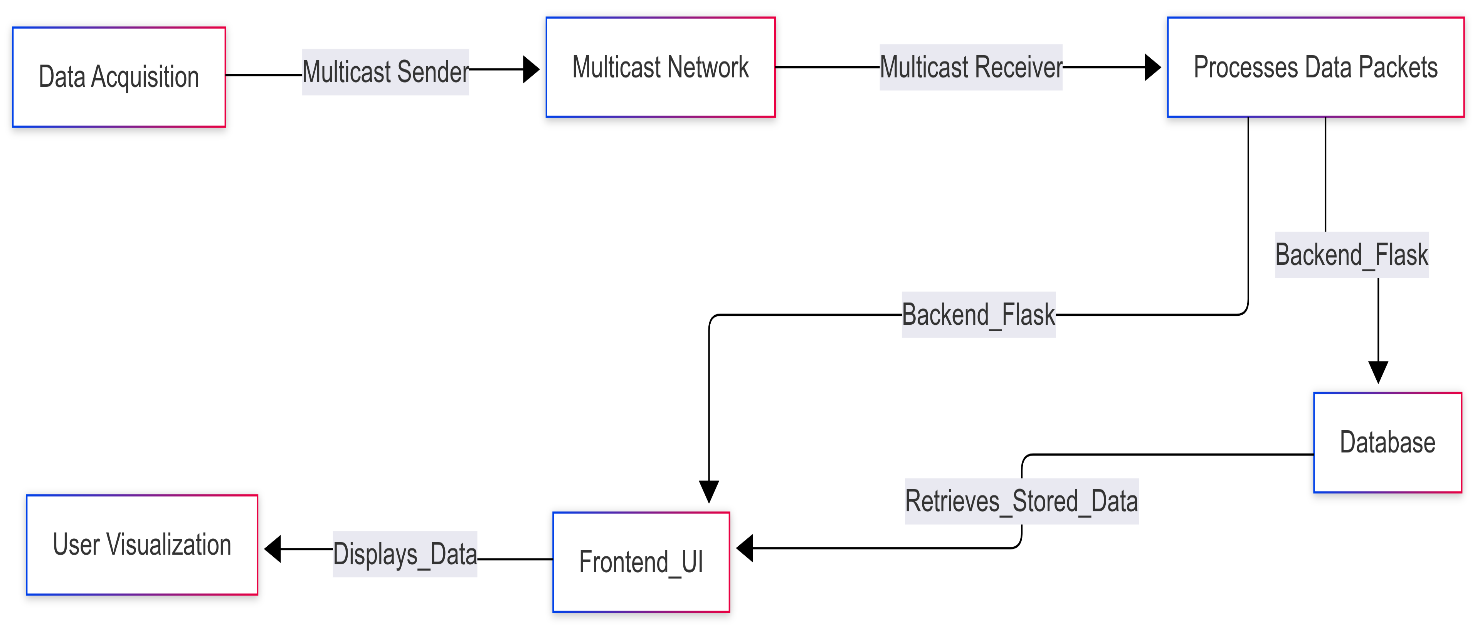
The NavIC Ranging Data Monitoring System is composed of four major components:

* **Multicast Sender (Data Generation):**
  + Reads binary log files (e.g., for Station 1 and Station 2).
  + Transmits NavIC data via UDP multicast.
* **Multicast Receiver (Data Logging):**
  + Listens to specific multicast groups.
  + Logs incoming binary data into station-specific log files.
* **Backend Server (Flask Application – Processing & API):**
  + Processes the logged binary files.
  + Decodes range measurement data () and navigation data ().
  + Provides a REST API endpoint to supply processed data to the frontend.
* **Frontend (HTML, JavaScript – UI for Visualization):**
  + Fetches data via the API.
  + Dynamically displays range measurement data and navigation logs in tabular format.



**Figure 2 Multicast Data Flow**

## Data Flow Explanation

* **Step 1:** The **Multicast Sender** reads pre-recorded binary log files and continuously transmits data over designated UDP multicast groups.
* **Step 2:** The **Multicast Receiver** subscribes to these multicast groups and writes the received data into log files (one for each station, such as Station 1 and Station 2).
* **Step 3:** The **Backend Server** (implemented using Flask) runs multiple threads—each monitoring a log file for a specific station. It decodes the binary data:
  + For **range measurement data ():** The backend extracts parameters like pseudo range, Doppler, carrier-to-noise ratio, etc.
  + For **navigation data ():** The backend processes the navigation sub frame details.
* **Step 4:** The **Frontend** periodically polls the backend API, retrieves the latest decoded data, and updates the UI, allowing users to switch channels (e.g., Station 1 or Station 2) to view respective data.

**Figure 3 Data Flow Diagram**

# NavIC Data Handling & Binary Log Decoding

## NavIC Log Structure

NavIC logs consist of a **binary header** and a variable-length payload. Key components include:

* **Sync Word:** A unique 4-byte pattern (e.g., 0xAACC4756) that identifies the beginning of a log message.
* **Message Length & ID:** These fields specify the total length of the message and the type for range measurement data and for navigation data.
* **Log Count, Time Status, Week, and Milliseconds:** These fields capture timing and sequencing information.
* **CRC:** A 32-bit cyclic redundancy check to ensure data integrity.

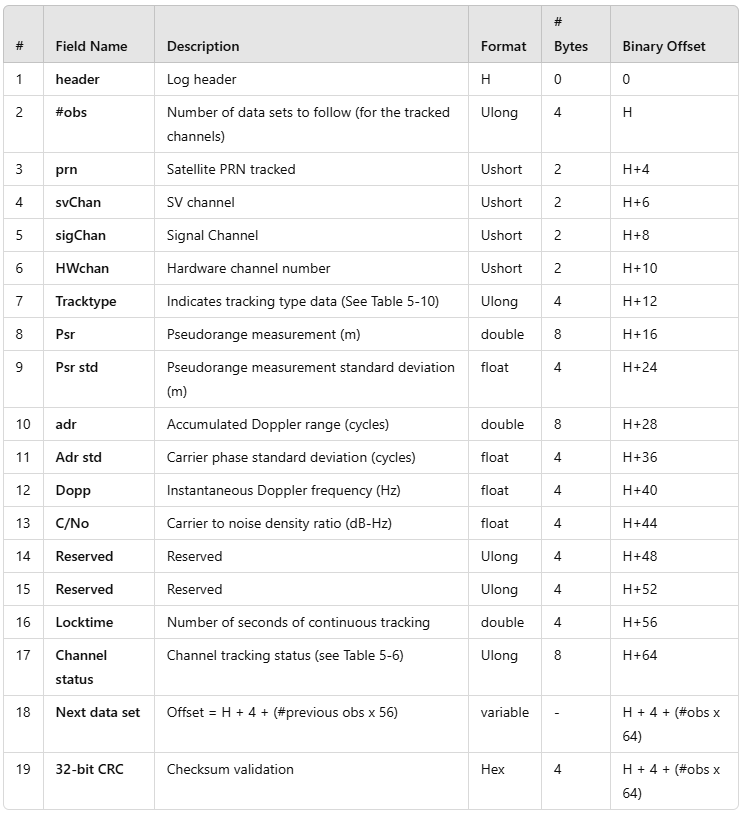
## Decoding Process for Range Measurement Data ()

* The decoding function reads the **header** and unpacks it using the Python struct module.
* **Range measurement Data Fields:**
  + **Observations Count:** Indicates the number of satellite observation records.
  + Each observation record (64 bytes) contains parameters like PRN, SV channel, signal channel, hardware channel, pseudo range (PSR), PSR standard deviation, Doppler frequency, carrier-to-noise ratio (C/No), lock time, and channel status.
* The decoded data is stored in a buffer indexed by channel (e.g., “Station 1” or “Station 2”).

## Decoding Process for Navigation Data ()

* Similar to range measurement data, the navigation data is decoded from its binary form.
* **Navigation Fields:**
  + Includes fields such as signal channel, hardware channel, PRN, signal type, parity status, parity failures, and raw data bits/bytes.
* The backend appends these records to a buffer for the corresponding channel.

## Handling Data Fragmentation

* For larger data packets, the NavIC system uses **fragmentation**:
  + A data packet is divided into multiple fragments if it exceeds a certain size.
  + Each fragment contains part of the data along with a fragment number.
* The backend logic uses the **sequence number** and **fragment number** from the header to reassemble the complete data message.

**Figure 4 NavIC Log Structure**

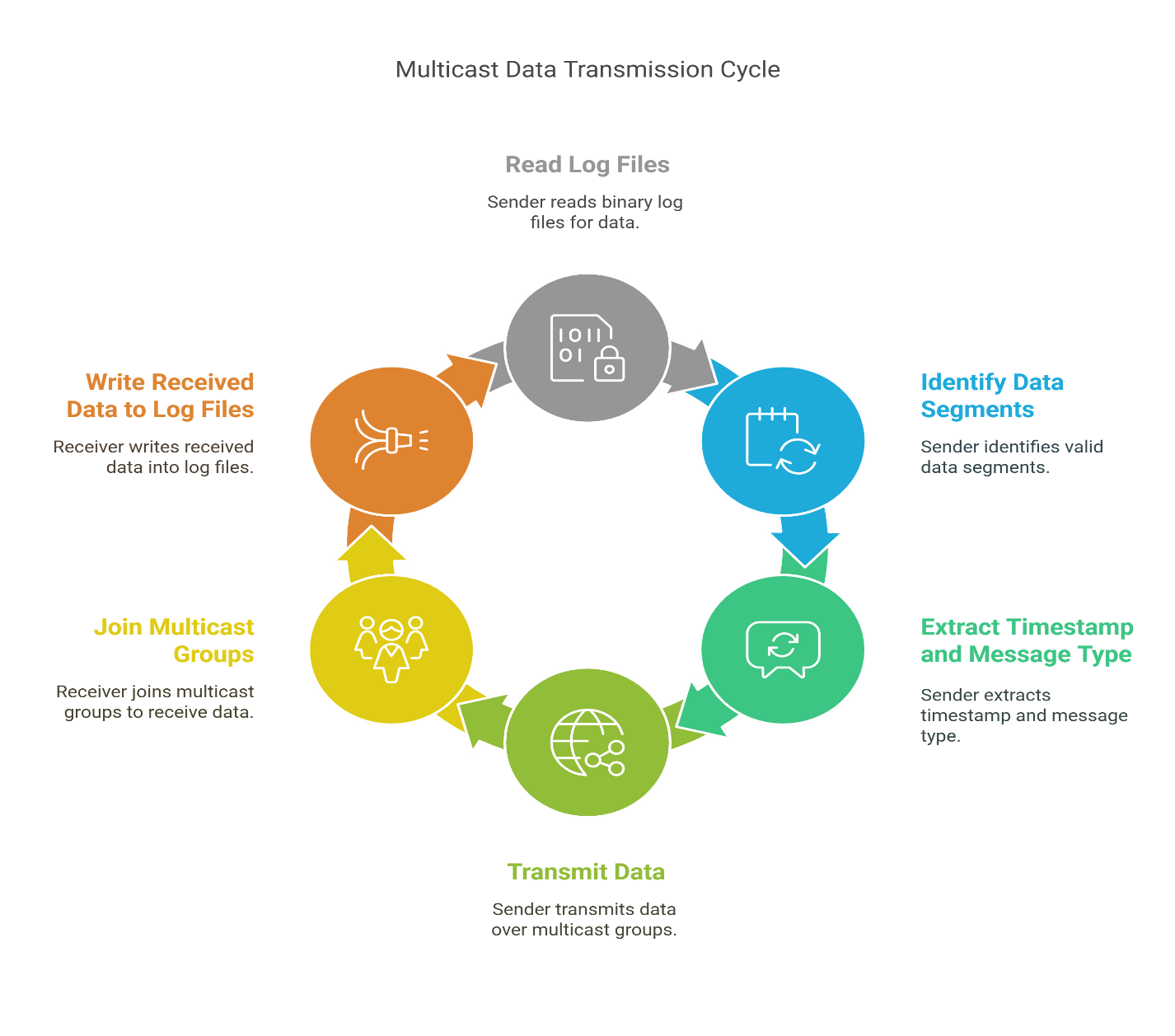
# Implementation Details

## Multicast Sender – daps\_mcast\_sender.py

* **Functionality:**
  + Reads binary log files (e.g., Station1\_daps\_log\_065.dat or Station2\_daps\_log\_065.dat).
  + Identifies valid data segments using sync words.
  + Extracts timestamp and message type.
  + Transmit data over two multicast groups for Station 1 and Station 2.
* **Key Considerations:**
  + **Timestamp Synchronization:** Ensures that data is sent in real time.
  + **Fragmentation Handling:** Properly transmits fragmented data as per NavIC protocol.

## Multicast Receiver – multicast\_rx.py

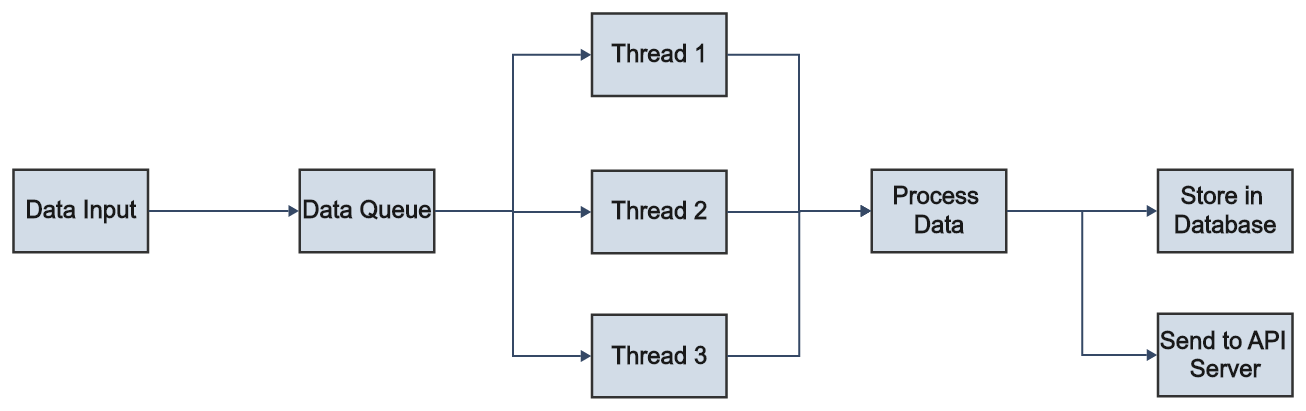
* **Functionality:**
  + Joins multicast groups to receive data.
  + Uses UDP sockets with appropriate options (e.g., setting TTL correctly with an unsigned byte).
  + Writes received data into station-specific log files (e.g., Station 1\_daps\_log\_065.dat and Station 2\_daps\_log\_065.dat).
* **Key Correction:**
  + TTL value is correctly packed using pack ('B', 255) instead of a signed byte.



**Figure 5 Multicast Sender-Receiver Workflow**

## Backend Server – app.py

* **Structure:**
  + Uses **Flask** to serve a web interface and provide REST API endpoints.
  + Launches separate file-reading threads for different channels (e.g., one for Station 1 and one for Station 2).
* **Data Processing:**
  + Each thread reads its assigned log file, decodes range measurement and navigation data, and updates global buffers.
  + The /fetch-data endpoint returns the latest processed data for the selected channel.
* **Error Handling:**
  + Ensures that data for non-active channels is not displayed on the UI.



**Figure 6 Backend Data Processing**

## Frontend – index.html and script.js

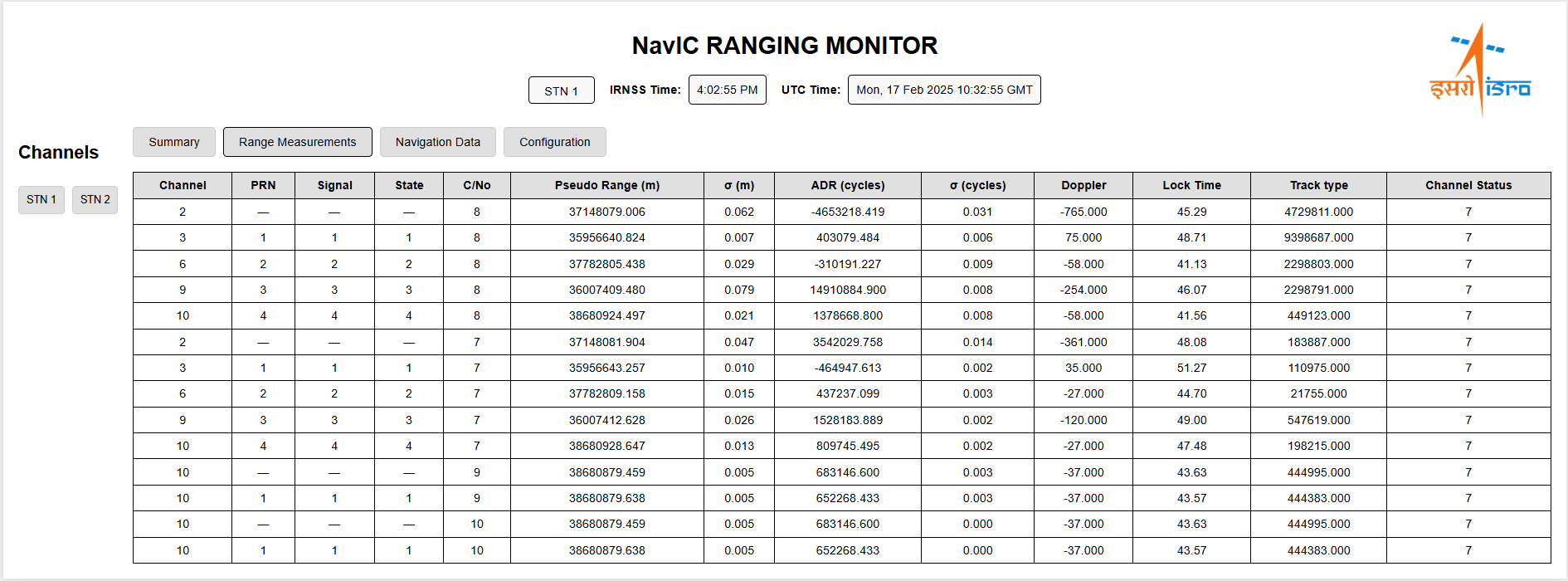
* **User Interface:**
  + Presents channel buttons allowing users to select different stations.
  + Contains tabs for displaying **Summary**, **Range measurements**, **Navigation Data**, and **Configuration**.
* **Dynamic Updates:**
  + Uses JavaScript to periodically poll the Flask API.
  + Updates the UI tables based on the fetched data.
* **Channel-Specific Data Handling:**
  + When a channel (e.g., Station 1 or Station 2) is selected, only its data is shown; any residual data from another channel is cleared.

# Testing and Results

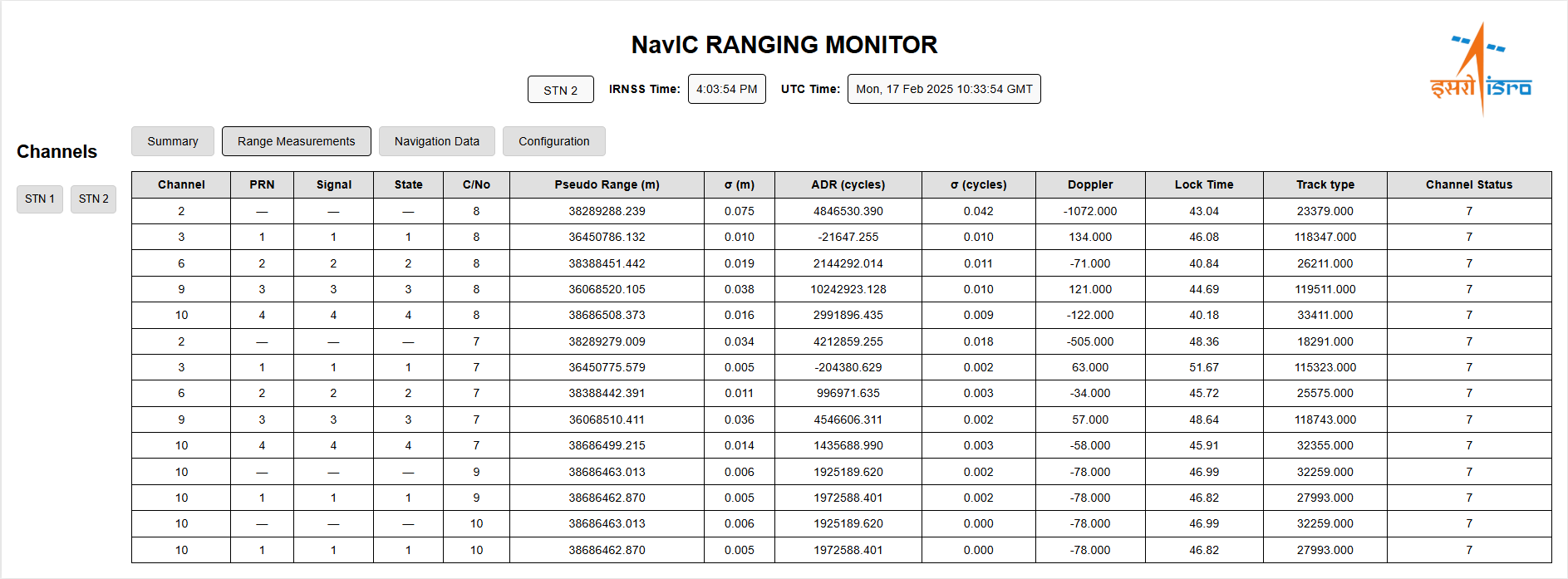
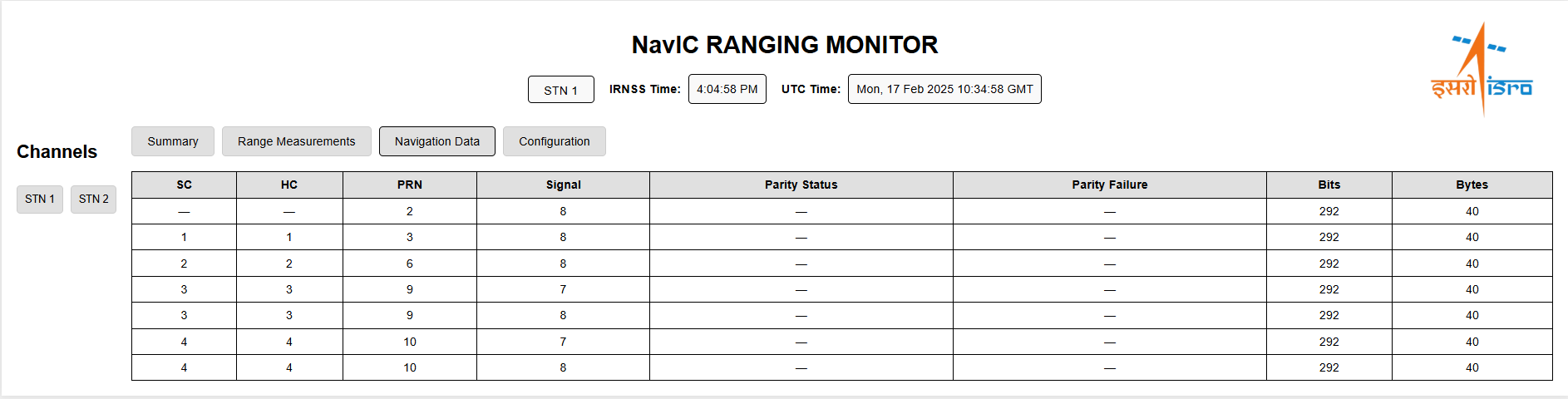
## Testing Methodology

* **Test Environment:**
  + Deployed the system on a test network with simulated NavIC data for both Station 1 and Station 2.
  + Utilized two separate terminals for data sender and receiver, and one for the backend server.
* **Test Cases:**
  1. **Channel Verification:**
     + Verify that clicking on "Station 1" displays only Station 1 data.
     + Verify that clicking on "Station 2" displays only Station 2 data.
  2. **Data Integrity:**
     + Compare the decoded parameters (pseudo range, Doppler, C/No, etc.) with expected values.
  3. **Real-Time Updates:**
     + Confirm that the UI updates every second with new data.
  4. **Fragmentation Handling:**
     + Test with large data packets to ensure that fragmentation and reassembly work as intended.

## Results Analysis

* **Successful Data Reception:**
  + Both Station 1 and Station 2 channels received and displayed the correct range measurement and navigation data.
* **UI Behaviour:**
  + The web interface correctly cleared data when switching channels, ensuring no residual data from the previous channel.
* **Log Decoding Accuracy:**
  + Backend logs confirmed the correct decoding of both range measurement and navigation data.
* **Performance Metrics:**
  + The system’s latency and update frequency met the real-time requirements.

**Figure 7 Station 1 Range measurement Data**



**Figure 8 Station 1 Navigation Data**

**Figure 9 Station 2 Range measurement Data**

## Debugging and Iteration

**Figure 10 Station 2 Navigation Data**

* **Issues Encountered:**
  + Initially, Station 1 data was displayed for Station 2 channel selection due to shared buffer handling.
  + Multicast receiver TTL packing error (resolved by switching from signed to unsigned byte).
* **Resolutions Implemented:**
  + Modified buffer management in the frontend.
  + Updated multicast receiver code to use pack ('B', 255).

# Challenges and Solutions

## Multi-Channel Processing

* **Challenge:**
  + Processing data from both Station 1 and Station 2 in a single backend instance without cross-contamination.
* **Solution:**
  + Implemented multi-threading in the Flask backend (app.py) to handle separate log files for each channel.
  + Ensured that API endpoints correctly filtered data by channel.

## Multicast Reception Issues

* **Challenge:**
  + Incorrect TTL value packaging resulted in errors during multicast reception.
* **Solution:**
  + Updated the multicast\_rx.py file to pack the TTL value as an unsigned byte pack ('B', 255).

## Data Fragmentation and Reassembly

* **Challenge:**
  + Handling fragmentation in data packets to ensure full message reassembly.
* **Solution:**
  + Utilized sequence and fragment numbers from the NavIC header for proper reassembly.

# Future Scope

## Expanding Station Support

* **Objective:**
  + Extend the system to support additional NavIC stations beyond Station 1 and Station 2.
* **Approach:**
  + Scale the backend by adding more threads for additional log files.

## Database Integration

* **Objective:**
  + Transition from in-memory buffers to a persistent database.
* **Benefits:**
  + Enables historical data analysis and long-term storage.
* **Potential Databases:**
  + PostgreSQL, MongoDB, or any other NoSQL database.

## Advanced Visualization

* **Objective:**
  + Enhance the user interface with advanced data visualization tools.
* **Approach:**
  + Incorporate charts, graphs, and heat maps for deeper insights.
  + Use libraries such as D3.js or Chart.js for dynamic visualizations.

## AI-Based Anomaly Detection

* **Objective:**
  + Integrate machine learning models to automatically detect anomalies or signal degradations.
* **Approach:**
  + Train models on historical NavIC data.
  + Trigger alerts when abnormal patterns are detected.

# Conclusion

## Summary of Achievements

* Developed a **real-time NavIC Ranging Data Monitoring System** capable of processing and displaying data from multiple stations.
* Implemented robust **multicast data transmission and reception** mechanisms.
* Achieved accurate **binary log decoding** for both range measurement and navigation data.
* Successfully integrated a **Flask-based backend** with a dynamic **web-based frontend**.

## Impact and Future Enhancements

* The system provides a scalable platform for real-time satellite monitoring.
* Future enhancements include support for additional stations, database integration, advanced visualizations, and AI-based analytics to further improve system performance and reliability.

# References

1. **ISRO Documentation:**
   * EICD for NavIC Ranging Reference Receiver.
   * IRNSS SIGNAL-IN-SPACE ICD.
2. **Technical References:**
   * Python’s struct module documentation.
   * Flask framework documentation.