INTERNATIONAL TELECOMMUNICATION UNION

**TELECOMMUNICATION STANDARDIZATION SECTOR**

STUDY PERIOD 2022-2024

**FOCUS GROUP ON AI NATIVE NETWORKS**

**AINN-I-xx**

**Original: English**

**Question(s):** N/A Virtual, TBD 2024

**INPUT DOCUMENT**

**Source:** *Indian Institute of Information Techonology, Allahabad*

**Title:** *IIITA_ECE Team - Report on ITU WTSA Hackathon 2024 – Eyes from Above*

**Contact:** Ram E-mail: iec2021019@iiita.ac.in

**Abstract:** This document contains the submission of a report for IIITA_ECE Team towards ITU WTSA Hackathon 2024 for use case *Eyes from Above.*

# Use case introduction: “Eyes from Above”

**Namami Gange Programme** is an Integrated Conservation Mission, approved as a Flagship Programme by the Union Government of India in June 2014 with a budget outlay of ₹22,500 crore from 2023–26 to accomplish the twin objectives of effective abatement of pollution, conservation and rejuvenation of National River Ganga[1]. One of the serious problems that this project faced is waste accumulation on the river bed which in turn is getting into the river. If we could identify the waste accumulation on the river bed and proactively take steps for it not to get into the river stream, it will benefit the programme.

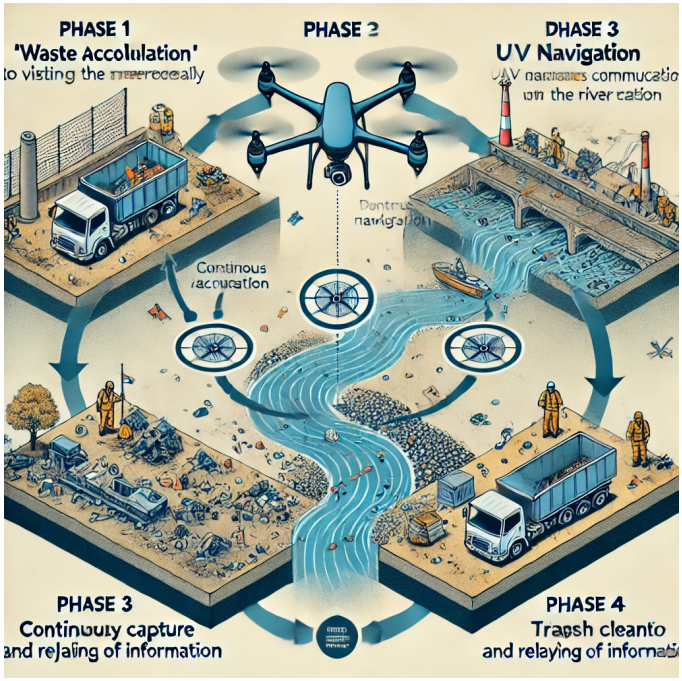
But **Ganga River** is spread across a vast area with having maximum length and width of approximately 1,543 km and 1024 km respectively. The total length of river Ganga (measured along the Bhagirathi and the Hooghly) up to its outfall into Bay of Bengal is 2,525 km with 631 km navigable length[2].

Surveilling all that area manually is not possible. Thus to enable the act of surveillence for waste accumulation across the river or altleast along a part of it, I am developing this “Eyes from Above”.

It utilizes UAV(Unmanned Aerial Vehicle)’s to monitor the area and send the location information for immediate waste collection.

*Waste was accumulated on the river bed because of human actions or evironmental factors. Then our UAV will be deputed to do a scheluded inspection along that segment of the river.*
*UAV will be navigated along the riverbed and send the images back for processing.*
*The images can be processed to figure out the extent of waste accumulation if any. Prioritized areas will be immediately cleaned to prevent the waste from reaching into the river.*

Consider the scene map below:




"AI generated pic from DallE"

Phase 1: "*Waste Accumulation"*: Scheduled UAV visits on the river bed.

Phase 2: "*UAV navigation"*: UAV is released and navigated across the river bed while in constant communication with base station.

Phase 3: "*Continuous capture and relaying of information"*: Image information is relayed onto the computing station for analysing the presence of trash.

Phase 4: "*River bed cleaning"*: Based on the presence of trash, personnel can be sent to clean the specific part of the river bed.

# Use case requirements

Requirement-1: It is critical that path of the UAVis planned accordingly.

Requirement-2: It is critical that Machine Learning models should be trained with sufficent data to enable the smooth handling of edge case scenarios.

Requirement-3: It is critical that sensors on-board of the UAV are relaying correct information.

Requirement-4: It is critical to normalize and re-distribute the resources to all available areas for better overall efficiency of the programme.

# PS1: Pipeline Design

- Several AI/ML concepts are used.

1) River Network Identification [3]

- Classification Algorithms are applied on recent satellite images to find out the path of the river.
- Supervised machine learning with algorithms like Support Vector Machine and Random Forest have shown promising results.

2) Path Planning
- Reinforcement Learning algorithms have been shown to be efficient in traning UAV's to respond by themselves.[4]
- SkyScenes dataset can be used to train the UAVs.[5]

3) Object Detection
- Deep convolutional neural networks have shown to perform well in differentiating the background with waste found on ground.
- Images with UAV vision angle can be procured from the **UAVVaste Dataset** [6]

## System Design:

**1. Non-RT RIC:**

- **Role**: Non-RT RIC provides long-term network management and training for AI/ML models that will be used in the xApp. In your case, it will:
  - Train the **river detection** and **waste detection** models using historical data (satellite images and previous flight data).
  - Optimize the models periodically and update them on Near-RT RIC when improvements are made.
- **Tasks**:
  - Maintain training datasets (satellite images, waste site identification).
  - Develop policies for drone operation and update them periodically based on performance data.

**2. Near-RT RIC:**

- **Role**: Near-RT RIC hosts the xApp, interacts with the RAN components in real-time, and makes immediate decisions based on the data provided by the drone and the network.
- **Components**:
  - **xApp**: This is the core application that handles both drone navigation and waste detection.
  - **ML Models**: Integrated in the xApp, including:
    - **River Detection Model**: This is the pre-built model that processes satellite images and identifies river edges.
    - **Waste Detection Model**: An object detection model that analyzes images captured by the drone and identifies waste disposal sites.
- **Tasks**:
  - Receive and process satellite images to identify the river paths using the **River Detection Model**.
  - Coordinate drone movement along the river path using 5G connectivity, ensuring it stays close to the river's edge.
  - Collect real-time data from the drone (e.g., camera feeds, GPS locations) and analyze the feed to detect waste using the **Waste Detection Model**.

- Log locations where waste is detected and relay the information to external systems or for further analysis.

**3. Drone:**

- **Role**: The drone is the physical entity that follows the path provided by the xApp and collects real-time data. The drone communicates with the RAN (through gNB/eNB) using 5G/4G for reliable and high-speed data transfer.
- **Tasks**:
  - Follow the river edge path by receiving commands from the xApp hosted on Near-RT RIC.
  - Capture high-resolution images as it flies along the river.
  - Send the images back to the xApp for real-time processing.
  - Upon detecting waste, log the location and capture additional data (images/videos) as needed.

**4. RAN Nodes (gNB/eNB):**

- **Role**: These are the base stations responsible for providing 5G/4G connectivity to the drone in the field. They ensure continuous communication between the drone and the xApp running on the Near-RT RIC.
- **Tasks**:
  - Provide low-latency connectivity to the drone for command/control signals.
  - Enable high-bandwidth data transmission for real-time image/video streams.
  - Support mobility management as the drone moves along the river.

**5. Satellite Image Processing:**

- **River Path Detection**: The satellite images are processed by the ML engine integrated into the xApp. This model analyzes the image to identify the river path.
- **Coordinate Generation**: Based on the image analysis, the river path is converted into a series of GPS coordinates that the drone will follow.

**6. Waste Detection:**

- As the drone follows the path, the xApp continuously analyzes the drone's camera feed using the **Waste Detection Model**. When waste is detected:
  - The xApp logs the location (using the drone's GPS) and saves relevant images.
  - This data can be sent back to the Non-RT RIC for further analysis or reporting.

## Requirements

- SRC of data: Camera
- Collector: edge server
- Models: River Network Identification, Path planning, Object detection
- Policies: Smart Navigation
- Distributors : Edge server, Cloud Processing
- Model inference Application (SINK): xApp


"AI generated pic from DallE”

System YAML Code:

```
tosca_definitions_version: tosca_simple_yaml_1_3

# Define custom node types for the xApp, UAV, ML models, and
personnel dispatch systems
node_types:
  xapp_type:
    derived_from: tosca.nodes.SoftwareComponent
    interfaces:
      Standard:
        inputs:
          river_path_data:
            value: { get_input: river_path_data }
            type: string
          waste_site_coordinates:
            value: { get_input: waste_site_coordinates }
            type: string
          operations:
            create: playbooks/create_xapp.yaml
            delete: playbooks/delete_xapp.yaml

  ml_model_type:
    derived_from: tosca.nodes.SoftwareComponent
    interfaces:
      Standard:
```

```
        inputs:
          satellite_image_feed:
            value: { get_input: satellite_image_feed }
            type: string
          operations:
            create: playbooks/create_ml_model.yaml
            delete: playbooks/delete_ml_model.yaml

  drone_controller_type:
    derived_from: tosca.nodes.SoftwareComponent
    interfaces:
      Standard:
        inputs:
          drone_commands:
            value: { get_input: drone_commands }
            type: string
          operations:
            create: playbooks/create_drone_controller.yaml
            delete: playbooks/delete_drone_controller.yaml

  personnel_dispatch_type:
    derived_from: tosca.nodes.SoftwareComponent
    interfaces:
      Standard:
        inputs:
          cleanup_instruction:
            value: { get_input: cleanup_instruction }
            type: string
          operations:
            create: playbooks/create_dispatch.yaml
            delete: playbooks/delete_dispatch.yaml

# Define inputs for the entire topology
topology_template:
  inputs:
    satellite_image_feed:
      type: string
      default: "Satellite_Image_Data"
    river_path_data:
      type: string
      default: "River_Path_Data"
    waste_site_coordinates:
      type: string
      default: "Waste_Site_Coordinates"
    drone_commands:
      type: string
      default: "Drone_Commands"
    cleanup_instruction:
      type: string
      default: "Personnel_Dispatch_Instruction"

  # Define the node templates (components) for the system
```

```
  node_templates:
    my-workstation:
      type: tosca.nodes.Compute
      attributes:
        private_address: localhost
        public_address: localhost

    river_path_ml:
      type: ml_model_type
      requirements:
        - host: my-workstation
      interfaces:
        Standard:
          create: playbooks/create_ml_model.yaml
          delete: playbooks/delete_ml_model.yaml

    waste_detection_ml:
      type: ml_model_type
      requirements:
        - host: my-workstation
      interfaces:
        Standard:
          create: playbooks/create_ml_model.yaml
          delete: playbooks/delete_ml_model.yaml

    drone_controller:
      type: drone_controller_type
      requirements:
        - host: my-workstation
      interfaces:
        Standard:
          create: playbooks/create_drone_controller.yaml
          delete: playbooks/delete_drone_controller.yaml

    xapp:
      type: xapp_type
      requirements:
        - host: my-workstation
      interfaces:
        Standard:
          create: playbooks/create_xapp.yaml
          delete: playbooks/delete_xapp.yaml

    personnel_dispatch:
      type: personnel_dispatch_type
      requirements:
        - host: my-workstation
      interfaces:
        Standard:
          create: playbooks/create_dispatch.yaml
          delete: playbooks/delete_dispatch.yaml
```

```
# Outputs to display relevant data after the xApp operation
outputs:
  xapp_output:
    description: "Results from xApp operations."
    value: { get_attribute: [ xapp, public_address ] }
```

# PS2: xApp design

- xApp is designed to handle the intricasies of the complex system and manage all the moving parts.
- ML models are implemented at different regions to balance the load at computing end.
- Drone commands are processed in single step at a time

**xApp design**:
Code:

```
import socket
import time
from datetime import datetime
from real_time_processing import process_river_path, path_planning,
detect_waste_sites  # Assumes pre-built ML models
from non_real_time_processing import get_river_path
# Command dictionary for controlling the drone
cmds = {
    'START_MISSION': 'DRONE_START_MISSION',
    'STOP_MISSION': 'DRONE_STOP_MISSION',
    'NEXT_STEP' : 'DRONE_CALIBRATE_DIRECTION',
    'TAKE_IMAGE': 'DRONE_TAKE_IMAGE',
    'MARK_LOCATION': 'DRONE_MARK_LOCATION',
    'MOVE_TO_NEXT': 'DRONE_MOVE_TO_NEXT_POINT',
    'RETURN_BASE': 'DRONE_RETURN_TO_BASE'
}

class XApp:
    def __init__(self, ip, port):
        self.conn = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.conn.connect((ip, port))
        self.river_path = None
        self.drone_status = "IDLE"

    def send_request(self, message):
       """Send a message to the RAN node or drone to request data or
issue commands."""
        self.conn.send(f"{message} at {datetime.now().strftime('%H:
%M:%S')}".encode('utf-8'))

    def receive_data(self):
        """Receive image or I/Q data from the RAN node."""
        data = self.conn.recv(16384)
        if data:
            print(f"Received data at {datetime.now().strftime('%H:%M:
%S')}")
```

```
            return data
        return None

    def process_data(self, data):
        """Run ML algorithms and  on received data to detect river
paths."""
        if process_river_path(data):
            self.river_path = data  # Store detected river path
            print("River path identified.")
            self.next_step = path_planning(data)
            print("Next Step computed.")
        else:
            print("River path not detected, retrying...")

     # Send images to Non-RealTime Processing to check for waste and
marking location
        if self.river_path:
            waste_detected = detect_waste_sites(self.river_path)
            if waste_detected:
                print("Waste sites detected along river.")
                self.send_request(cmds['MARK_LOCATION'])
                return True
            else:
                print("No waste detected.")
        return False

    def run(self):
        """Main loop to run the xApp and control the drone."""
        while True:
            # Start mission
            self.send_request(cmds['START_MISSION'])

            # Receive satellite image or sensor data
            data = self.receive_data()
            if data:
                # Process the data using ML models
                if self.process_data(data):
                    # If waste is detected, move drone to next location
                    self.send_request(cmds['MOVE_TO_NEXT'])
                else:
                    # No waste detected, stop the mission or retry
                    self.send_request(cmds['RETURN_BASE'])

            time.sleep(1)  # Sleep before checking for next data

            # You can add conditions to stop the mission
            if self.drone_status == "RETURNED":
                break

    def stop(self):
        """Stop the xApp."""
        self.send_request(cmds['STOP_MISSION'])
```

```
        self.conn.close()

if __name__ == "__main__":
    xapp = XApp(ip="192.168.1.1", port=8080)
    try:
        xapp.run()
    except KeyboardInterrupt:
        xapp.stop()
```

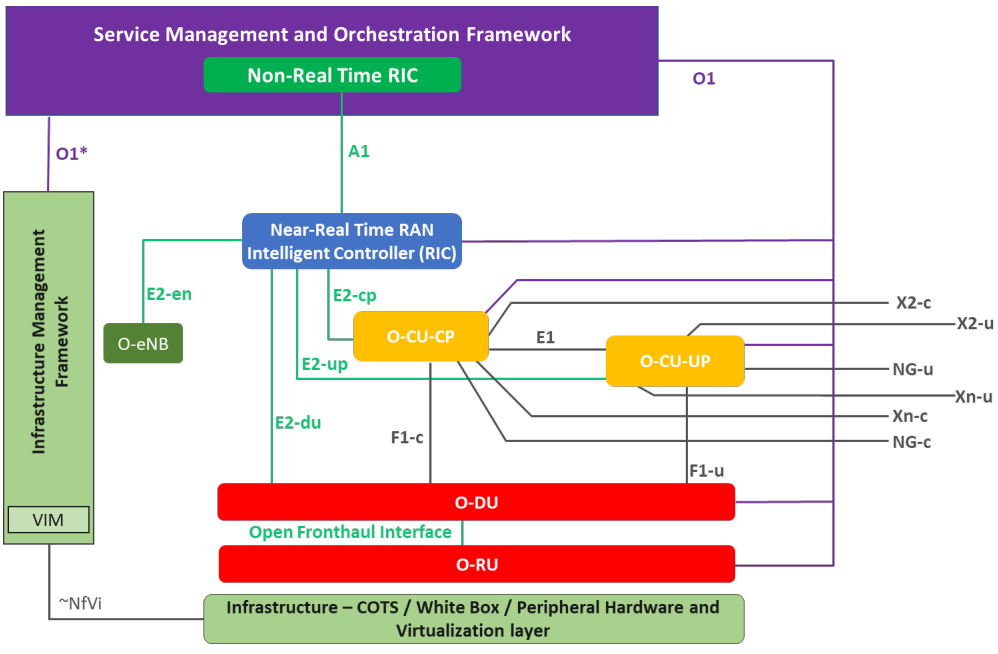
# Relation to Standards

**ITU-T Y.3061 (Autonomous Networks)**: This standard provides an architecture framework for autonomous networks, focusing on requirements, architecture components, and interactions between these components. For your drone-based system, adhering to Y.3061 would mean incorporating clear architectural components such as:

- **Autonomous control modules**: These would handle the decision-making for drone navigation and waste site detection.
- **Self-management features**: The drone should have capabilities to adapt to changing environmental conditions (e.g., river paths) without human intervention.
- **Sequence diagrams**: Your xApp should have well-defined interactions between the ML model for river path detection and drone navigation systems.

**ITU-T Y.3072 (Information-Centric Networking in IMT-2020)**: This standard is critical for efficient data handling, particularly in low-latency, scalable environments such as drone navigation over large areas. Your system can benefit from Y.3072 by:

- **Name Mapping and Resolution**: Efficient handling of large datasets (e.g., satellite images, river path coordinates) through named data objects would allow the xApp to fetch required data quickly without delay.
- **Scalability and low-latency requirements**: Given that drones need real-time data on river paths and waste disposal sites, the network must be optimized to deliver timely and accurate information.



# References:

[1] https://www.india.gov.in/spotlight/namami-gange-programme#tab=tab-1
[2] https://indiawris.gov.in/downloads/Ganga%20Basin.pdf
[3] Seetha, M., Lalitha Parameswari, D.V., Malini Devi, G. (2023). River Network Identification from Satellite Imagery Using Machine Learning Algorithms. In: Seetha, M., Peddoju, S.K., Pendyala, V., Chakravarthy, V.V.S.S.S. (eds) Intelligent Computing and Communication. ICICC 2022. Advances in Intelligent Systems and Computing, vol 1447. Springer, Singapore. https://doi.org/10.1007/978-981-99-1588-0_32
[4] Wang, J., Zhao, Z., Qu, J. *et al.* APPA-3D: an autonomous 3D path planning algorithm for UAVs in unknown complex environments. *Sci Rep* **14**, 1231 (2024). https://doi.org/10.1038/s41598-024-51286-2
[5] *Khose, Sahil, Anisha Pal, Aayushi Agarwal, Deepanshi, Judy Hoffman, Prithvijit Chattopadhyay and Georgia Tech. "SkyScenes: A Synthetic Dataset for Aerial Scene Understanding." ArXiv abs/2312.06719 (2023): n. pag.*
[6] Kraft, Marek, Mateusz Piechocki, Bartosz Ptak, and Krzysztof Walas. 2021. "Autonomous, Onboard Vision-Based Trash and Litter Detection in Low Altitude Aerial Images Collected by an Unmanned Aerial Vehicle" *Remote Sensing* 13, no. 5: 965. https://doi.org/10.3390/rs13050965